This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2022.3215752

Efficient and Generic Algorithms for Quantitative Attack Tree Analysis

Milan Lopuhaä-Zwakenberg[†] Carlos E. Budde[*] Mariëlle Stoelinga[†‡]
[*]University of Trento, CyberSecurity, Trento, Italy.
[†]University of Twente, Formal Methods and Tools, Enschede, the Netherlands.
[‡]Radboud University, Department of Software Science, Nijmegen, the Netherlands.
carlosesteban.budde@unitn.it {m.a.lopuhaa,m.i.a.stoelinga}@utwente.nl

Abstract—Numerous analysis methods for quantitative attack tree analysis have been proposed. These algorithms compute relevant security metrics, i.e. performance indicators that quantify how good the security of a system is; typical metrics being the most likely attack, the cheapest, or the most damaging one. However, existing methods are only geared towards specific metrics or do not work on general attack trees. This paper classifies attack trees in two dimensions: proper trees vs. directed acyclic graphs (i.e. with shared subtrees); and static vs. dynamic gates. For three out of these four classes, we propose novel algorithms that work over a generic attribute domain, encompassing a large number of concrete security metrics defined on the attack tree semantics; dynamic attack trees with directed acyclic graph structure are left as an open problem. We also analyse the computational complexity of our methods.

Index Terms—Attack trees, security metrics, BDD algorithms, computational complexity, formal methods.

1 Introduction

Attack trees (ATs) are important tools to analyse the security of complex systems. Their intuitive definition and general applicability makes them widely studied in academia and used in industry. ATs are part of many system engineering frameworks, e.g. *UMLsec* [1, 2] and *SysMLsec* [3], and supported by industrial tools such as Isograph's *AttackTree* [4].

An AT is a hierarchical diagram that describes potential attacks on a system. Its root represents the attacker's goal, and the leaves represent basic attack steps: indivisible actions of the attacker. Intermediate nodes are labeled with gates, that determine how their children activate them. The most basic notions of ATs have OR and AND gates only; many extensions exist to model more elaborate attacks.

Attack trees are often studied via *quantitative analysis*, where ATs are assigned a wide range of security metrics. Typical examples of such metrics are the minimal time [5, 6, 7, 8], minimal cost [9], or maximal probability [10] of a successful attack, as well as Pareto analyses that study trade offs among attributes [5, 11]. Calculating such metrics is essential when comparing alternatives or making trade offs. This leads to the following research problem:

We thank Sebastiaan Joosten for his help with the proof of Theo. 5.2, and Lars Kuijpers for collaborations that led to Algo. 3. This work was partially supported by ERC Consolidator Grant 864075 (CAESAR). Funded by the European Union under GA n.101067199-ProSVED. Views and opinions expressed are those of the author(s) only and do not necessarily reflect those of the European Union or The European Research Executive Agency. Neither the European Union nor the granting authority can be held responsible for them.

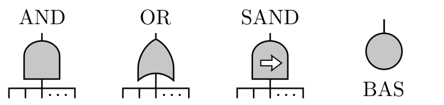


Figure 1: Nodes in an attack tree.

Question. *How can AT metrics be calculated efficiently?*

Numerous algorithms have been proposed to address this question. Such algorithms exploit a plethora of techniques, for instance Petri nets [12], model checking [9], and Bayesian networks [13]. While these algorithms provide good ways to compute metrics, they also suffer from several drawbacks: (1) Many of them are geared to specific attributes, such as attack time or probability, and the procedure may not extend to other metrics; (2) Several algorithms do not exploit the acyclic structure of the AT, especially approaches based on model checking; (3) Since their application is mostly illustrated on small examples, it is unclear how these approaches scale to larger case studies.

The aim of this work is to answer the question above for a general class of metrics. The answer hinges on two factors:

1) *Static vs. dynamic ATs:* Apart from the standard ATs, which we call *static*, an important extension are *dynamic* ATs. These allow for sequential-AND (SAND) gates, which require their children to succeed in left-to-right order [10, 9]. A formal approach to analyse dynamic ATs requires more rich semantics than for static ATs.
2) *Tree vs. DAG structure:* Their name notwithstanding, ATs can be directed acyclic graphs (DAGs), in which a node may have multiple parents. The additional condition that an AT is tree-structured allows for considerably faster computation of metrics.

Thus, our contribution answers the question as follows:

Answer. We provide efficient and generic algorithms to compute AT metrics, by tailoring them to our 2-dimensional

© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: UNIVERSITA TRENTO. Downloaded on December 12,2022 at 12:12:57 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2022.3215752

Metric	Static tree	Dynamic tree	Static DAG	Dynamic DAG
min cost	BU [14, 15, 16]	BU [10]	MTBDD [17] $\mathcal{C}$-BU [18]	PTA [5]
min time	BU [14, 19]	APH [6] BU [10]	Petri nets [12]	MILP [8]
min skill	BU [14, 20]	BU [10]	$\mathcal{C}$-BU [18]	—
max damage	BU [14, 19, 20]	BU [10]	MTBDD [17] DPLL [21]	PTA [5]
probability	BU [22, 19]	APH [6]	BDD [23] DPLL [21]	I/O-IMC [9]
Pareto fronts	BU [24, 19]	**Lemma 10.4**	$\mathcal{C}$-BU [11] **Lemma 10.4**	PTA [5]
Any of the above	**Algo. 1:** $\texttt{BU}_{\texttt{SAT}}$	**Algo. 4:** $\texttt{BU}_{\texttt{DAT}}$	**Algo. 2:** $\texttt{BDD}_{\texttt{DAG}}$	**OPEN PROBLEM**[1]
***k*-top metrics**	BU-projection [14]	BU [25] **Lemma 10.3**	**Algo. 3:** `BDD_shortest_paths`	**OPEN PROBLEM**[2]

Table 1: Efficient algorithms to compute security metrics on different AT classes (details and abbreviations are in Sec. 11). [1] Algo. 5 reduces runtime for any found method; [2] Lemma 10.3 reduces k-top calculation to metric calculation.

categorisation: static vs. dynamic ATs, and tree-structured vs. DAG-structured ATs.

On page 3 we give a precise description of our contributions, presented in accordance to the categorisation above. Our algorithmic results are summarised in Table 1, and Sec. 11 provides an elaborate comparison with related work.

There are various naming conventions in the AT literature. We follow the terminology of works like [14, 17, 26, 27], akin to fault tree standards on which attack trees were inspired [28]. Two points to highlight in this respect are our use of *attack tree* to refer also to DAG-like structures, and our use of *dynamic* to mean ATs with sequential-AND gates. These and all other terms used in this work are formalised in Secs. 2.2, 3.1 and 3.2, and Secs. 6.1 to 6.3. In Sec. 11 we compare our terminology to alternatives in the literature.

Static trees: The simplest category of Attack Trees are tree-structured static ATs. As shown in a seminal paper by Mauw & Oosdijk [14], metrics can be computed for these ATs in a bottom-up fashion, using appropriate operators $\triangledown$ and $\vartriangle$ on a set V, resp. for the OR and AND gates in the tree. We show this as Algo. 1. A key insight in [14] is that this procedure works whenever the algebraic structure $(V, \vartriangle, \triangledown)$ constitutes a semiring, i.e. $\vartriangle$ must distribute over $\triangledown$.
We provide an alternative proof of correctness for this result: while [14] deploys rewriting rules for attack trees, we show in Sec. 9 that it directly follows from the validity of *modular analysis* on DAG-structured ATs. Furthermore, we propose new categories of attribute domains, which extend the application of the bottom-up algorithm to compute popular security metrics. Ordered semiring domains constitute an important category of metrics, for which we show that derived metrics such as Pareto fronts and k-top values also fall within the semiring attribute domain framework.

Static DAGs: It is well-known that static ATs with DAG structure cannot be studied with bottom-up procedures [23, 29]. Many algorithms exist to tackle such ATs, mostly geared to specific metrics [9, 17, 12, 21, 18]. Sec. 5 presents a generic algorithm that works for any semiring attribute domain $(V, \triangledown, \vartriangle)$ that is absorbing, i.e. $x \triangledown (x \vartriangle y) = x$, and has neutral elements $1_\triangledown$ and $1_\vartriangle$ for operators $\triangledown$ and $\vartriangle$ resp.

Concretely, we exploit a binary decision diagram representation (BDD) of the attack tree. Our algorithm visits each BDD node once and is thus linear in its size. The caveat is that BDDs can be of exponential size in the number of basic attack steps (BASes), but one cannot hope for faster algorithms: as we show, computing a minimal attack is an NP-hard problem. However, in practice the BDD approach still yields computational gains: we show that we can split up the calculation according to the *modules* of the AT, i.e. subDAGs only connected to the rest of the AT by their root. This speeds up calculations considerably. Moreover, BDDs are known to be compact in practice [30], and allow parallel traversals [31], making them an overall efficient choice. Furthermore, we show that the bottom-up algorithm—of linear runtime on any static tree—works also when applied to DAG ATs, if the operators $\triangledown$ and $\vartriangle$ are idempotent, i.e. $x \triangledown x = x \vartriangle x = x$.

Dynamic trees: Metrics for dynamic attack trees (DATs) are usually decoupled from semantics, and defined either on the syntactic AT structure, or ad hoc for the selected computation method [10, 5, 32, 18]. The main obstacle to a semantics-based approach is to choose semantics for DATs in a way that supports a proper definition of metric, i.e. that is compatible with the notion of metric of static ATs, and that is as generic as the attribute domains from [14]. In particular, the interaction among multiple SAND gates is nontrivial, because they may impose conflicting execution orders on the BAS of the tree.

We follow [26, 8] in giving semantics to DATs via partially ordered sets (*posets*). Each poset $\langle A, \prec\rangle$ represents an attack scenario, where A collects all attacks steps to be performed, and $a \prec b$ indicates that step a must be completed before step b starts. This set up enables us to define a notion of metric for DATs based on their semantics. *A key contribution is defining general metrics for dynamic attack trees, and showing that tree-structured DATs are analysable by extending the bottom-up algorithm with an additional operator (see Algo. 4).* Concretely, we use attribute domains with three operators: $\triangledown$, $\vartriangle$, $\vartriangleright$, where $\vartriangleright$ distributes over $\triangledown$ and $\vartriangle$, and $\vartriangle$ distributes over $\triangledown$. We prove this algorithm correct in our formal semantics. This is relevant and non-trivial because (a) earlier algorithms do not provide explicit correctness results in terms of semantics, and (b) the metrics are form-

© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: UNIVERSITA TRENTO. Downloaded on December 12,2022 at 12:12:57 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2022.3215752

ally defined on the poset semantics of a DAT, while the algorithm works on its syntactic AT structure.

Dynamic DAGs: Efficient computation of metrics for DAG-structured DATs is left as future research challenge. A naïve, inefficient algorithm would enumerate all posets in the semantics. Instead, one could extend BDD-algorithms for static DAGs to dynamic ATs. This is non-trivial, as BDDs ignore the order of attack steps. Thus, efficient analysis of DAG-structured DATs is an important open problem. Yet we do show how modular analysis works here as well, and can be used to speed up any algorithm.

Paper structure: We list our contributions next, and give minimal background in Sec. 2. Secs. 3 to 5 study static attack trees, and Secs. 6 to 8 study dynamic attack trees. Sec. 9 presents modular analysis, and Sec. 10 studies Pareto fronts and k-top metrics. The paper discuses related work and concludes in Secs. 11 and 12.

Contributions of [26]: An earlier version of this paper was published in [26], whose key contributions are:

1. An efficient and generic BDD-based algorithm for DAG-SATs, working for absorbing semiring attribute domains with neutral elements (Sec. 5);
2. A theorem proving that computing a minimal successful attack is NP-hard (Sec. 5.1);
3. The adaptation of semiring attribute domains to DATs, defining general metrics on DATs (Sec. 6);
4. A bottom-up algorithm for tree-structured DATs (Sec. 7);
5. A BDD-based algorithm to compute k-top metrics on SATs (Sec. 5.5);
6. Future directions to analyse DAG-DATs efficiently (identified as an open problem in Sec. 8).

New contributions in this version: Contributions of the current paper over [26] are:

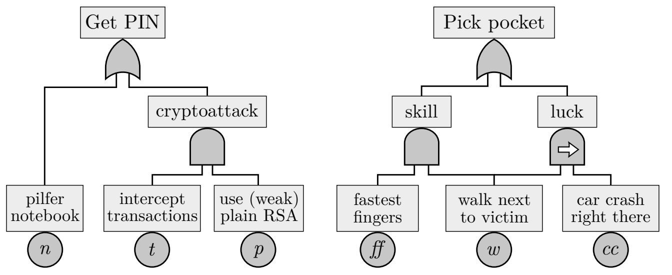
7. The result that for idempotent semiring domains, bottom-up algorithms—of linear runtime—work even for DAG-structured SATs (Sec. 5.2);
8. A unified method to calculate multiple metrics simultaneously—as needed by Pareto fronts, k-top metrics, and uncertainty sets—exploiting ordered attribute domains (Sec. 10);
9. General semantics for DATs that cover the entire universe of models, also less restrictive than [26] for the well-formedness criterion (Sec. 6.2);
10. A formalised modular analysis technique to improve the runtime of DAG algorithms, by combining analysis results for independent subtrees (Sec. 9);
11. An improved description of the BDD-based algorithm, illuminating the intuition behind the method (Algo. 2 in Sec. 5.3).

We place ourselves in the literature in Table 1 and Sec. 11.

2 Attack Trees

2.1 Attack tree models

Syntactically, an attack tree is a rooted DAG that models an undesired event caused by a malicious party, e.g. a theft. ATs



(a) A static-tree AT: T_s (b) A dynamic-DAG AT: T_d

Figure 2: Attack tree models

show a top-down decomposition of a top-level attack—the root of the DAG—into simpler steps. The leaves are basic steps carried out by the attacker. The nodes between the basic steps and the root are intermediate attacks, and are labelled with gates to indicate how its input nodes (children) combine to make the intermediate attack succeed.

Basic Attack Steps: The leaves of the AT represent indivisible actions carried out by the attacker, e.g. smash a window, decrypt a file, etc. These BAS nodes can be enriched with attributes, such as its execution time, the cost incurred, and the probability with which the BAS occurs. We model attributes via an attribution function $\alpha\colon \text{BAS} \to V$.

Gates: Non-leaf nodes serve to model intermediate steps that lead to the top-level goal. Each has a logical *gate* that describes how its children combine to make it succeed: an OR gate means that the intermediate attack will succeed if any of its child nodes succeeds; an AND gate indicates that all children must succeed, in any order or possibly in parallel; a SAND gate (viz. sequential-AND) needs all children to succeed sequentially in a left-to-right order.

Example 2.1. Fig. 2a shows a static attack tree, T_s, that models how a PIN code can be obtained by either pilfering a notebook, or via a cryptographic attack. The pilfering is considered atomic, while the cryptoattack consists of two steps which must both succeed: intercepting transactions, and abusing weak RSA encryption. Note that T_s has a plain tree structure. Instead, Fig. 2b shows a dynamic attack tree, T_d, with a DAG structure. Its TLA is to pick a pocket, which is achieved either by having "skill" or "luck." In both cases the attacker must walk next to the victim, so these gates share the BAS child w, making T_d not a tree. In the case of "luck" the order of events matters: if the attacker first walks next to the victim and then a traffic accident happens, the pick-pocket succeeds. Thus, this intermediate attack is modelled with a SAND gate. Instead, "fastest fingers" is an inherent attacker flair that is always present. It is thus meaningless to speak of an order w.r.t. the attacker-victim encounter, so an AND gate is used.

Security metrics: A key goal in quantitative security analysis is to compute relevant *security metrics*, which quantify how well a system performs in terms of security. Typical examples are the cost of the cheapest attack, the probability of the most likely one, the damage produced by the most harmful one, and combinations thereof. Security metrics for ATs are typically obtained by combining the attribute values $\alpha(a) \in V$ assigned to each $a \in \text{BAS}$. For instance, the

© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: UNIVERSITA TRENTO. Downloaded on December 12,2022 at 12:12:57 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2022.3215752

cheapest attack is that for which the sum of the cost of its BASes is minimal. *The topic of this paper is how to compute large classes of security metrics in generic and efficient ways.*

2.2 Attack tree syntax

ATs are rooted DAGs with typed nodes: we consider types $\mathbb{T} = \{\mathtt{BAS}, \mathtt{OR}, \mathtt{AND}, \mathtt{SAND}\}$. The edges of an AT are given by a function ch that assigns to each node its (possibly empty) sequence of children. We use set notation for sequences, e.g. $e \in (e_1, \ldots, e_m)$ means $\exists i.\, e_i = e$, and we denote the empty sequence by ε.

Definition 2.2. An *attack tree* is a tuple $T = (N, t, ch)$ where:

- N is a finite set of *nodes*;
- $t\colon N \to \mathbb{T}$ gives the *type* of each node;
- $ch\colon N \to N^*$ gives the sequence of *children* of a node.

Moreover, T satisfies the following constraints:

- (N, E) is a connected DAG, where $$E = \{(v,u) \in N^2 \mid u \in ch(v)\};$$
- T has a unique root, denoted R_T: $$\exists!\, R_T \in N.\, \forall v \in N.\, R_T \notin ch(v);$$
- BAS_T nodes are the leaves of T: $$\forall v \in N.\, t(v) = \mathtt{BAS} \Leftrightarrow ch(v) = \varepsilon.$$

We omit the subindex T if no ambiguity arises, e.g. an attack tree $T = (N, t, ch)$ defines a set $\mathrm{BAS} \subseteq N$ of basic attack steps. If $u \in ch(v)$ then u is called a *child* of v, and v is a *parent* of u. We write $v = \mathrm{AND}(v_1, \ldots, v_n)$ if $t(v) = \mathtt{AND}$ and $ch(v) = (v_1, \ldots, v_n)$, and analogously for OR and SAND, denoting $ch(v)_i = v_i$. Moreover we denote the universe of ATs by $\mathscr{T}$ and call $T \in \mathscr{T}$ *tree-structured* if $\forall v, u \in N.\, ch(v) \cap ch(u) = \varepsilon$; else we say that T is *DAG-structured*. If $\mathtt{SAND} \notin t(N)$ we say that T is a *static attack tree* (SAT); else it is a *dynamic attack tree* (DAT).

3 Static Attack Trees

In the absence of SAND gates the order of execution of the BASes is irrelevant. This allows for simple semantics given in terms of a Boolean function called *structure function*. The computation of security metrics, however, crucially depends on whether the AT is tree- or DAG-structured.

3.1 Semantics for static attack trees

The semantics of a SAT is defined by its successful attack scenarios, in turn given by its structure function. First, we define the notions of attack and attack suite.

Definition 3.1. An *attack scenario*, or shortly an *attack*, of a static AT T is a subset of its basic attack steps: $A \subseteq \mathrm{BAS}_T$. An *attack suite* is a set of attacks $\mathcal{S} \subseteq 2^{\mathrm{BAS}_T}$. We denote by $\mathscr{A}_T = 2^{\mathrm{BAS}_T}$ the universe of attacks of T, and by $\mathscr{S}_T = 2^{2^{\mathrm{BAS}_T}}$ the universe of attack suites of T. We omit the subscripts when there is no confusion.

Intuitively, an attack suite $\mathcal{S} \in \mathscr{S}$ represents different ways in which the system can be compromised. From those, one is interested in attacks $A \in \mathcal{S}$ that actually represent a threat. For instance for T_s in Example 2.1 one such attack is $\{t, p\}$. In contrast, $\{t\}$ is an attack that does not succeed, i.e. it does not reach the top-level goal. The structure function $f_T(v, A)$ indicates whether the attack $A \in \mathscr{A}$ succeeds at node $v \in N$ of T. For Booleans we use $\mathbb{B} = \{1, 0\}$.

Definition 3.2. The *structure function* $f_T\colon N \times \mathscr{A} \to \mathbb{B}$ of a static attack tree T is given by:

$$f_T(v, A) = \begin{cases} 1 & \text{if } t(v) = \mathtt{OR} \text{ and } \exists u \in ch(v).\, f_T(u, A) = 1, \\ 1 & \text{if } t(v) = \mathtt{AND} \text{ and } \forall u \in ch(v).\, f_T(u, A) = 1, \\ 1 & \text{if } t(v) = \mathtt{BAS} \text{ and } v \in A, \\ 0 & \text{otherwise.} \end{cases}$$

An attack A is said to *reach* a node v if $f_T(v, A) = 1$, i.e. it makes v succeed. If no proper subset of A reaches v, then A is a *minimal attack on* v. The set of attacks reaching v is denoted Suc_v, and the set of minimal attacks on v is denoted $[\![v]\!]$. We define $f_T(A) \doteq f_T(R_T, A)$, and attacks that reach R_T are called *succesful*. We write $[\![T]\!] \doteq [\![R_T]\!]$ and $\mathrm{Suc}_T \doteq \mathrm{Suc}_{R_T}$. Furthermore, the minimal attacks on R_T (i.e. the minimal succesful attacks) are called *minimal attacks*. The minimal attacks relate to the structure function in the sense that

$$f_T(v, A) = \bigvee_{A' \in [\![v]\!]} \bigwedge_{a \in A'} \delta_a(A),$$

where $\delta_a(A) = 1$ iff $a \in A$. This can also be applied the other way around: we can represent T by a propositional formula L_T by replacing each $a \in \mathrm{BAS}$ with the atom δ_a, and each node with its corresponding logical connector. Then the conjunctions in the minimal disjunctive normal form of L_T correspond to the minimal attacks on T.

Definition 3.3. For $v \in N$, the propositional formula $L_{T,v}$ with atoms in $\{\delta_a \mid a \in \mathrm{BAS}_T\}$ is given by

$$L_{T,v} = \begin{cases} \bigvee_{w \in ch(v)} L_{T,w} & \text{if } t(v) = \mathtt{OR}\ , \\ \bigwedge_{w \in ch(v)} L_{T,w} & \text{if } t(v) = \mathtt{AND}, \\ \delta_v & \text{if } t(v) = \mathtt{BAS}. \end{cases}$$

Furthermore, we define $L_T := L_{T,R_T}$.

SATs are *coherent* [33], meaning that adding attack steps preserves success: if A is successful then so is $A \cup \{a\}$ for any $a \in \mathrm{BAS}$. Thus, the suite of successful attacks of an AT is characterised by its minimal attacks. This was first formalised in [14], and is called multiset semantics in [34]:

Definition 3.4. The *semantics of a SAT T* is its suite of minimal attacks $[\![T]\!]$.

Example 3.5. The SAT in Example 2.1, T_s (Fig. 2a), has three successful attacks: $\{n\}$, $\{t, p\}$, and $\{n, t, p\}$. The first two are minimal, so we have: $[\![T_s]\!] = \{\{n\}, \{t, p\}\}$.

An alternative characterisation of this semantics for tree-structured SATs is shown as Lemma 3.6, which also provides the key argument for correctness of the bottom-up procedure (Algo. 1 in Sec. 4). For tree-structured SATs, Lemma 3.6 can be used to compute the semantics of Def. 3.4 by recursively applying cases *1)–3)* to $[\![R_T]\!] = [\![T]\!]$. However, BDD representations provide more compact encodings of this semantics (see Sec. 5).

We formulate Lemma 3.6 for binary ATs; its extension to arbitrary trees is straightforward but notationally cumbersome. The proof of Lemma 3.6 is given in Appendix A.

© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: UNIVERSITA TRENTO. Downloaded on December 12,2022 at 12:12:57 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2022.3215752

Lemma 3.6. *Consider a SAT with nodes $a \in BAS, v_1, v_2 \in N$. Then:*

1) $[\![a]\!] = \{\{a\}\}$;
2) $[\![OR(v_1, v_2)]\!] \subseteq [\![v_1]\!] \cup [\![v_2]\!]$;
3) $[\![AND(v_1, v_2)]\!] \subseteq \{A_1 \cup A_2 \mid A_1 \in [\![v_1]\!] \wedge A_2 \in [\![v_2]\!]\}$;
4) *All RHS of cases 1–3 consist of succesful attacks.*

If T is tree-structured then furthermore:

5) *In cases 2 and 3 the $[\![v_i]\!]$ are disjoint, and in case 3 moreover the A_i are disjoint;*
6) *Equality holds in cases 2 and 3.*

3.2 Security metrics for static attack trees

Lemma 3.6 allows for *qualitative analyses*, i.e. finding the successful attacks of minimal size. To enable *quantitative analyses*, i.e. computing security metrics such as the minimal time and cost among all attacks, all BASes are enriched with attributes. We define security metrics in three steps: first an attribution α assigns a value to each BAS; then a security metric $\widehat{\alpha}$ assigns a value to each attack scenario; and finally the metric $\check{\alpha}$ assigns a value to each attack suite.

Definition 3.7. Given an AT T and a set V of values:

1) an *attribution* $\alpha\colon \mathrm{BAS}_T \to V$ assigns an *attribute value* $\alpha(a)$, or shortly an *attribute*, to each basic attack step a;
2) a *security metric* refers both to a function $\widehat{\alpha}\colon \mathscr{A}_T \to V$ that assigns a value $\widehat{\alpha}(A)$ to each attack A;
and to a function $\check{\alpha}\colon \mathcal{S}_T \to V$ that assigns a value $\check{\alpha}(\mathcal{S})$ to each attack suite $\mathcal{S}$.

We write $\check{\alpha}(T)$ for $\check{\alpha}([\![T]\!])$, setting the metric of an AT to the metric of its suite of minimal attacks.

Example 3.8. Let $V = \mathbb{N}$ denote time, so that $\alpha(a)$ gives the time required to perform the basic attack step a. Then the time needed to complete an attack A can be given by $\widehat{\alpha}(A) = \sum_{a\in A} \alpha(a)$, and the time of the fastest attack in a suite $\mathcal{S}$ is $\check{\alpha}(\mathcal{S}) = \min_{A\in\mathcal{S}} \widehat{\alpha}(A)$. If instead $V = [0,1] \subseteq \mathbb{R}$ denotes probability, then the probability of an attack is given by $\widehat{\alpha}(A) = \prod_{a\in A} \alpha(a)$, and the probability of the likeliest attack in a suite is $\check{\alpha}(\mathcal{S}) = \max_{A\in\mathcal{S}} \widehat{\alpha}(A)$.

Def. 3.7 gives a very general notion of metric. For a more operational definition—which enables computation for static ATs, but does not depend on their tree/DAG-structure—one must resort to the semantics. For this we follow an approach along the lines of Mauw and Oostdijk [14]. Namely, we define a *metric function* $\check{\alpha}\colon \mathscr{T} \to V$ that yields a value for each SAT based on its semantics, an attribution, and two binary operators $\triangledown$ and $\vartriangle$.

Definition 3.9. Let V be a set:

1) an *attribute domain* over V is a tuple $D = (V, \triangledown, \vartriangle)$, whose *disjunctive operator* $\triangledown\colon V^2 \to V$, and *conjunctive operator* $\vartriangle\colon V^2 \to V$, are associative and commutative;
2) the attribute domain is a *semiring*[1] if $\vartriangle$ distributes over $\triangledown$, i.e. $\forall\, x, y, z \in V.\ x \vartriangle (y \triangledown z) = (x \vartriangle y) \triangledown (x \vartriangle z)$;
3) let T be a static AT and α an attribution on V. The *metric for $\mathcal{S}$* associated to α and D is given by:[2]

$$\check{\alpha}(\mathcal{S}) \;=\; \underbrace{\mathop{\bigtriangledown}_{A\in\mathcal{S}}}_{\check{\alpha}} \; \underbrace{\mathop{\bigtriangleup}_{a\in A}}_{\widehat{\alpha}} \alpha(a).$$

4) The metric for T associated to α and D is given by $\check{\alpha}(T) \doteq \check{\alpha}([\![T]\!])$.

Example 3.10. The fastest attack time metric from Example 3.8 comes from the semiring attribute domain $(\mathbb{N}, \min, +)$; indeed, the time for an attack is the sum of the attack times for all constituting BASes ($\vartriangle = +$), and the attack time for the AT is the time of the fastest attack ($\triangledown = \min$). Similarly, the highest attack probability comes from the semiring attribute domain $([0,1], \max, \cdot)$. Consider the SAT $T_s = \mathrm{OR}\big(n, \mathrm{AND}(t,p)\big)$ from Fig. 2a. These two metrics can be calculated as follows.

1) *Fastest attack*: Recall that $[\![T_s]\!] = \{\{n\},\{t,p\}\}$, and consider an attribution $\alpha = \{n \mapsto 1, t \mapsto 100, p \mapsto 0\}$. Then:

$$\begin{aligned}\check{\alpha}(T_s) =&\ \widehat{\alpha}(\{n\}) \triangledown \widehat{\alpha}(\{t,p\})\\ =&\ \alpha(n) \triangledown \big(\,\alpha(t) \vartriangle \alpha(p)\big)\\ =&\ \min(1, 100+0) \;=\; 1.\end{aligned}$$

2) *Most probable attack:* Now consider an attribution $\alpha' = \{n \mapsto 0.07, p \mapsto 0.01, t \mapsto 0.95\}$ for the same tree:

$$\begin{aligned}\check{\alpha}'(T_s) &= \alpha'(n) \triangledown' \big(\,\alpha'(t) \vartriangle' \alpha'(p)\big)\\ &= \max(0.07, 0.95\cdot 0.01) \;=\; 0.07.\end{aligned}$$

4 Computations for tree-structured SATs

Example 3.10 illustrates how to compute metrics for SATs using Def. 3.9. However, this method requires to first compute the semantics of the attack tree, which is *exponential* in the number of nodes $|N|$—see Theo. 5.2 or [18].

A key result in [14] is that metrics defined on attribute domains $(V, \triangledown, \vartriangle)$ that are semirings, can be computed via a bottom-up algorithm that is *linear* in $|N| + |E|$ (assuming constant time complexity of $\triangledown$ and $\vartriangle$) as long as the static AT has a proper tree structure. We repeat this result here, giving a more direct proof of correctness. We extend the result to dynamic attack trees in Sec. 5.

4.1 Bottom-up algorithm

First we formulate the procedure as Algo. 1, which propagates the attribute values from the leaves of the SAT to its root, interpreting OR gates as $\triangledown$ and ANDs as $\vartriangle$. This algorithm is linear in $|N| + |E|$ since each node v in the tree T is visited once, and at v we have $\deg(v) - 1$ computation steps. Algo. 1 can be called on any node of T: to compute the metric $\check{\alpha}(T)$ it must be called on its root node R_T.

We state the correctness of Algo. 1 in Theo. 4.1, which we prove in Appendix D. This result was proven in [14] via rewriting rules for ATs with a slightly different structure denoted "bundles". Our result concerns attack trees in the

1. Since we require $\vartriangle$ to be commutative, D is in fact a commutative semiring. Rings often include a neutral element for disjunction and an absorbing element for conjunction, but these are not needed in Def. 3.9.

2. This expression motivates our notation $\widehat{\alpha}$ and $\check{\alpha}$, which are similar to $\vartriangle$ and $\triangledown$, respectively. These notations, in turn, were chosen in [14] to be similar to $\wedge$ and $\vee$, respectively; the connection between these operators and the logical connectors is expressed in Theo. 4.1.

© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: UNIVERSITA TRENTO. Downloaded on December 12,2022 at 12:12:57 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2022.3215752

```
Input: Static attack tree $T = (N, t, ch)$,
       node $v \in N$,
       attribution $\alpha$,
       semiring attribute domain $D = (V, \triangledown, \vartriangle)$.
Output: Metric value $\check{\alpha}([\![v]\!]) \in V$.
if $t(v) = \texttt{OR}$ then
    return $\bigtriangledown_{u \in ch(v)} \texttt{BU}_{\texttt{SAT}}(T, u, \alpha, D)$
else if $t(v) = \texttt{AND}$ then
    return $\bigtriangleup_{u \in ch(v)} \texttt{BU}_{\texttt{SAT}}(T, u, \alpha, D)$
else // $t(v) = \texttt{BAS}$
    return $\alpha(v)$
```

Algorithm 1: $\texttt{BU}_{\texttt{SAT}}$ for a tree-structured SAT T

syntax from Def. 2.2, which is more conforming to the broad literature [15, 16, 22, 21, 35, 17, 5].

Theorem 4.1. *Let T be a static AT with tree structure, α an attribution on V, and $D = (V, \triangledown, \vartriangle)$ a semiring attribute domain. Then $\check{\alpha}(T) = \texttt{BU}_{\texttt{SAT}}(T, R_T, \alpha, D)$.*

4.2 Metrics as semiring attribute domains

Many relevant metrics for security analyses on SATs can be formulated as semiring attribute domains. Table 2 shows examples, where $\mathbb{N}_\infty = \mathbb{N} \cup \{\infty\}$ includes 0 and ∞. For instance "min cost" can be formulated in terms of $(\mathbb{N}_\infty, \min, +)$, which is a semiring attribute domain because $+$ distributes over $\min$, i.e. $a + \min(b, c) = \min(a + b, a + c)$ for all $a, b, c \in \mathbb{N}_\infty$. Also, attribute domains can handle SAND gates if the execution order is irrelevant for the metric, which happens e.g. for min skill and max damage.

Besides the metrics of Table 2, one can also be interested in derived concepts, such as the Pareto front of multiple metrics, or uncertainty sets when the attribute values of BASes are unknown. We show that such concepts fit into the semiring attribute domain framework in Sec. 10.

Non-semiring metrics: However, some meaningful metrics do fall outside this category. For instance and as noted in [14], the cost to defend against all attacks is represented by $(\mathbb{N}_\infty, +, \min)$; but Algo. 1 cannot compute this metric because $\min$ does not distribute over $+$. Less well-known is that total attack probability—i.e. $\check{\alpha}(T) = \sum_{A \in \mathrm{Suc}_T} \widehat{\alpha}(A)$ where $\widehat{\alpha}(A) = \left(\prod_{a \in A} \alpha(a)\right) \cdot \left(\prod_{a \notin A}(1 - \alpha(a))\right)$—can neither be formulated as an attribute domain. The problem is that $\widehat{\alpha}(A)$ does not have the shape $\bigtriangleup_{a \in A} \alpha(a)$, and that the sum is taken over all succesful attacks rather than just the minimal ones. Interestingly though, this probability can still be computed via a bottom-up procedure by taking $\check{\alpha}(\mathrm{AND}(v_1, v_2)) = \check{\alpha}(\mathrm{Suc}_{v_1}) \cdot \check{\alpha}(\mathrm{Suc}_{v_2})$ and $\check{\alpha}(\mathrm{OR}(v_1, v_2)) = \check{\alpha}(\mathrm{Suc}_{v_1}) + \check{\alpha}(\mathrm{Suc}_{v_2}) - \check{\alpha}(\mathrm{Suc}_{v_1} \cap \mathrm{Suc}_{v_2})$.

Stochastic analyses: Semirings form a bicomplete category, so finite and infinite products exist [36]. This allows to propagate not only tuples of attribute values, but also functions over them. In particular, *cumulative density functions* that assign a probability $t \mapsto P[X \leqslant t]$ constitute a semiring [6]. Such functions serve e.g. to consider attack probabilities (and cost, and damage) as functions that evolve on time.

Absorbing semirings: Although in this paper we calculate metrics by considering all *minimal* attacks, one could also simply consider all attacks. For many metrics this does not make a difference: for example, the successful attack with minimal cost will always be a minimal attack, since adding BASes can only increase the cost. Therefore, in the calculation of min cost we may as well take the minimum over all successful attacks, rather than just minimal attacks. On the other hand, when calculating max damage one will get a different answer when taking all attacks into account, as the full attack of all BASes will do more damage than a smaller minimal attack. The difference between these metrics can be described mathematically as follows. We call a semiring attribute domain $D = (V, \triangledown, \vartriangle)$ *absorbing* if $x \triangledown (x \vartriangle y) = x$ for all $x, y \in V$. If D is absorbing and α is an attribution into D, then $\widehat{\alpha}(A) \triangledown \widehat{\alpha}(A') = \widehat{\alpha}(A)$ for any two attacks with $A \subseteq A'$. It follows that for absorbing semiring attribute domains one has $\check{\alpha}(T) = \check{\alpha}(\mathrm{Suc}_T)$. Note that all metrics in Table 2 are absorbing except for max challenge and max damage.

METRIC	V	$\triangledown$	$\vartriangle$
min cost	$\mathbb{N}_\infty$	min	$+$
min time (sequential)	$\mathbb{N}_\infty$	min	$+$
min time (parallel)	$\mathbb{N}_\infty$	min	max
min skill	$\mathbb{N}_\infty$	min	max
max challenge	$\mathbb{N}_\infty$	max	max
max damage	$\mathbb{N}_\infty$	max	$+$
discrete prob.	$[0,1]$	max	$\cdot$
continuous prob.	$\mathbb{R} \to [0,1]$	max	$\cdot$

Table 2: SAT metrics with semiring attribute domains

5 Computations for DAG-structured SATs

Attack trees with shared subtrees cannot be analysed via a bottom-up procedure on its (DAG) structure, as we illustrate next in Example 5.1. This is a classical result from fault tree analysis [37], rediscovered for attack trees e.g. in [18].

There are many methods to analyse DAG-structured ATs: see Table 1 for contributions over the last 15 years, including [9, 17, 12, 21, 18]. These methods are often geared to specific metrics, e.g. cost, time, or probability [22, 21, 6]. Others use general-purpose techniques of high complexity and low efficiency, such as model checking [12, 5].

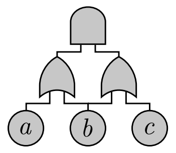
We present a novel algorithm based on a binary decision diagram (BDD) representation of the structure function of the SAT. BDDs offer a very compact encoding of Boolean functions [30], and are heavily used in model checking [38, 39, 40], as well as for probabilistic fault tree analysis [23, 41].

Our BDD-based approach works for absorbing semiring attribute domains, with neutral elements for operators $\triangledown$ and $\vartriangle$, regardless of the AT structure. It thus extends the generic and efficient result of [14]—that works for tree-structure SATs only—to DAG-structured SATs as well.

Our algorithm traverses the BDD bottom-up and it is linear in its size. However, BDDs can be exponential in the tree size [42]: but no asymptotically-faster algorithms exist, since the problem of computing metrics is NP-hard, as we show below. Moreover, BDDs are among the most efficient approaches in terms of performance of practical computation on Boolean formulae [30, 17].

© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: UNIVERSITA TRENTO. Downloaded on December 12,2022 at 12:12:57 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2022.3215752



Let: $\left.\begin{array}{ll} \alpha(a) = 3 & V = \mathbb{N}_\infty \\ \alpha(b) = 2 & \triangledown = \min \\ \alpha(c) = 4 & \vartriangle = + \end{array}\right\} D$

The cheapest attack is $\{b\}$: $\widehat{\alpha}(\{b\}) = 2$.

Figure 3: Metrics cannot be computed bottom-up on ATs with DAG structure. For min cost in this AT, Algo. 1 yields: $\mathtt{BU_{SAT}}(T, R_T, \alpha, D) = \min(3,2) + \min(2,4) = 4 \neq 2 = \check{\alpha}(T)$. The miscomputation stems from counting $\alpha(b)$ twice.

5.1 Computational complexity

We first show why the bottom-up procedure cannot compute metrics for ATs that have shared subtrees.

Example 5.1. Fig. 3 shows how the bottom-up approach can fail when applied to DAG-structured attack trees. Intuitively, the problem is that a visit to node v in Algo. 1—or any bottom-up procedure that operates on the AT structure—can only aggregate information on its descendants. So, the recursive call for v cannot determine whether a sibling node (i.e. any node v' which is not an ancestor nor a descendant of v) shares a BAS descendant with v. As a result, recursive computations for both v and v' may select a shared descendant $b \in \mathrm{BAS}$, and use $\alpha(b)$ in (both) their local computations. This causes the miscomputation in Fig. 3.

Known workarounds to this issue are keeping track of the BAS selected by the metric at each step [18, 29], and operating on the AT semantics [14]. In all cases the worst-case scenario has exponential complexity on the number of nodes of the attack tree: for [18] this is in the algorithm input, i.e. determining the sets of necessary and optional clones; instead for [14] and our Def. 3.9 the complexity lies in the computation of the semantics.

In general, one cannot hope for faster algorithms: Theo. 5.2 shows that the core problem—computing minimal attacks of DAG-structured attack trees—is NP-hard even in the simplest structure: plain attack trees with AND/OR gates. The proof (in Appendix E) reduces the satisfiability of logic formulae in conjunctive normal form, to the computation of minimal attacks in general SATs.

Theorem 5.2. *Given a DAG-structured static AT, the problem of computing any successful attack of minimal size is NP-hard.*

From this we can show that calculating metrics on DAG-structured SATs is NP-hard. To do this, we define the following attribute domain: for a SAT T, take $V = \mathcal{M}(\mathrm{BAS}_T)$, the set of multisets on BAS_T. Let $\vartriangle = \uplus$, i.e. multiset union. We identify $\mathcal{M}(\mathrm{BAS}_T) \cong \mathbb{N}^{\mathrm{BAS}_T}$, and $\uplus$ becomes $+$ under this identification. We furthermore choose an enumeration $\mathrm{BAS}_T = \{a_1, \ldots, a_n\}$, so that we may identify $V \cong \mathbb{N}^n$. We then define a map $\mu\colon V \to \mathbb{N}$ by

$$\mu(c) = \prod_{i=1}^{n} p_i^{c_i},$$

where p_i is the i-th prime; here an element $c \in V = \mathbb{N}^n$ is determined by its coefficients $c_i \in \mathbb{N}$. Define a linear order $\preceq$ on V by $c \preceq c'$ iff either $\sum_i c_i < \sum_i c'_i$, or $\sum_i c_i = \sum_i c'_i$ and $\mu(c) \leq \mu(c')$. Let $\triangledown = \min$ w.r.t. $\preceq$. One can then prove (see Appendix F) the following:

Lemma 5.3. *$(V, \triangledown, \vartriangle)$ is a semiring attribute domain. Furthermore, let $\alpha\colon BAS_T \to \mathcal{M}(BAS_T)$ be given by $\alpha(a) = \{\!\{a\}\!\}$. Then the multiset $\check{\alpha}(T)$ is a set, and it is the succesful attack of minimal size.*

With Theo. 5.2 this yields the following corollary:

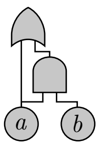
Corollary 5.4. *Computing a metric for a semiring attribute domain in a DAG-structured SAT is NP-hard.*

5.2 Idempotent semiring attribute domains

While in general computing metrics on DAG-structured SATs is hard, for some metrics the bottom-up algorithm still works. In Example 5.1 it is seen that $\mathtt{BU_{SAT}}$ fails because some BASes may be counted twice. However, when the operators $\triangledown, \vartriangle$ are such that multiple occurences of a BAS in a formula does not impact the calculation, then this is not a problem. One can express this formally by the following definition:

Definition 5.5 (Idempotent domain)**.** A binary operator $\star$ on a set X is called *idempotent* if $x \star x = x$ for all $x \in X$. A semiring attribute domain $D = (V, \triangledown, \vartriangle)$ is called idempotent if both operators $\triangledown$ and $\vartriangle$ are idempotent.

Idempotency of the domain is not enough for $\mathtt{BU_{SAT}}$ to work for DAG-structured SATs; we also need D to be absorbing. The reason for this is that in a DAG, there might be BASes that are not an element of any minimal attack, and hence are not present in the expression of $\check{\alpha}(T)$. Nevertheless, these BASes may still be used in the calculation of $\mathtt{BU_{SAT}}$, as in the following example.



Example 5.6. Consider the SAT to the right, $T = \mathrm{OR}(a, \mathrm{AND}(a, b))$, and consider the domain $D = (\mathbb{N}_\infty, \max, \max)$ for the max challenge metric; D is idempotent. Take $\alpha(a) = 1$, $\alpha(b) = 2$: since $[\![T]\!] = \{\{a\}\}$ then $\check{\alpha}(T) = \alpha(a) = 1$. However, Algo. 1 calculates $\mathtt{BU_{SAT}}(T, R_T, \alpha, D) = \max(\alpha(a), \max(\alpha(a), \alpha(b))) = 2$. The miscalculation comes from the fact that b is not an element of any minimal attack.

Only when D is both idempotent and absorbing, then $\mathtt{BU_{SAT}}$ calculates $\check{\alpha}(T)$ correctly. A motivating example is the domain $D = (\mathbb{N}_\infty, \min, \max)$, which represents the min time and min skill metrics; another example is $(\mathbb{B}, \vee, \wedge)$ from [18]. The fact that $\mathtt{BU_{SAT}}$ works for idempotent absorbing domains has been proven for the parallel min time metric in [8]; the theorem below extends this result to the general case. This result is similar to [18], where it is proven for Attack–Defense trees under mildly stronger assumptions on D, namely the existence of identity and absorbing elements in V.

Theorem 5.7. *Let T be a static AT, α an attribution on V, and $D = (V, \triangledown, \vartriangle)$ a semiring attribute domain. If D is idempotent and absorbing then $\check{\alpha}(T) = \mathtt{BU_{SAT}}(T, R_T, \alpha, D)$.*

5.3 Binary decision diagrams

BDDs offer a representation of Boolean functions that is often extremely compact. A BDD is a rooted DAG B_f that, intuitively, represents a Boolean function $f\colon \mathbb{B}^n \to \mathbb{B}$ over variables $Vars = \{x_i\}_{i=1}^n$. The terminal nodes of B_f

© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: UNIVERSITA TRENTO. Downloaded on December 12,2022 at 12:12:57 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2022.3215752

represent the outcomes of f: $\mathtt{0}$ or $\mathtt{1}$. A nonterminal node $w \in W$ represents a subfunction f_w of f via its Shannon expansion. That means that w is equipped with a variable $\mathit{Lab}(w) \in \mathit{Vars}$ and two children: $\mathit{Low}(w) \in W$, representing f_w in case that the variable $\mathit{Lab}(w)$ is set to $\mathtt{0}$; and $\mathit{High}(w)$, representing f_w if $\mathit{Lab}(w)$ is set to $\mathtt{1}$.

Definition 5.8. A *BDD* is a tuple $B = (W, \mathit{Low}, \mathit{High}, \mathit{Lab})$ over a set *Vars* where:

- The *set of nodes* W is partitioned into terminal nodes (W_t) and nonterminal nodes (W_n);
- $\mathit{Low}\colon W_n \to W$ maps each node to its *low child*;
- $\mathit{High}\colon W_n \to W$ maps each node to its *high child*;
- $\mathit{Lab}\colon W \to \{\mathtt{0},\mathtt{1}\} \cup \mathit{Vars}$ maps terminal nodes to Booleans, and nonterminal nodes to variables:
$$\mathit{Lab}(w) \in \begin{cases} \{\mathtt{0},\mathtt{1}\} & \text{if } w \in W_t, \\ \mathit{Vars} & \text{if } w \in W_n. \end{cases}$$

Moreover, B satisfies the following constraints:

- (W, E) is a connected DAG, where $E = \{(w, w') \in W^2 \mid w' \in \{\mathit{Low}(w), \mathit{High}(w)\}\}$;
- B has a unique root, denoted R_B: $\exists!\, R_B \in W.\, \forall w \in W_n.\, R_B \notin \{\mathit{Low}(w), \mathit{High}(w)\}$.

Given a BDD representing f, and $\boldsymbol{x} = (x_1, \ldots, x_n) \in \mathbb{B}^n$, one calculates $f(\boldsymbol{x})$ by starting from the root of the BDD, and at every node $w \in W_n$ with $\mathit{Lab}(w) = x_i$, proceed to $\mathit{High}(w)$ if $x_i = \mathtt{1}$, and to $\mathit{Low}(w)$ if $x_i = \mathtt{0}$. The terminal node one ends up in, is the value $f(\boldsymbol{x})$.

Reduced ordered BDDs: We operate with *reduced ordered BDDs*, simply denoted BDDs. This requires a total order $<$ over the variables. For Def. 5.8 this means that:

- *Vars* comes equipped with a total order, so B_f is actually defined over a pair $\langle \mathit{Vars}, < \rangle$;
- the variable of a node is of lower order than its children: $\forall\, w \in W_n.\ \mathit{Lab}(w) < \mathit{Lab}(\mathit{Low}(w)), \mathit{Lab}(\mathit{High}(w))$;
- the children of nonterminal nodes are distinct nodes;
- all terminal nodes are distinctly labelled;
- nonterminal nodes are uniquely determined by their label and children: $\forall w, w' \in W_n.\, (\mathit{Lab}(w) = \mathit{Lab}(w') \wedge \mathit{Low}(w) = \mathit{Low}(w') \wedge \mathit{High}(w) = \mathit{High}(w')) \Rightarrow w = w'$.

This has the following consequences in the BDD:

- there are exactly two terminal nodes: $W_t = \{\bot, \top\}$, with $\mathit{Lab}(\bot) = \mathtt{0}$ and $\mathit{Lab}(\top) = \mathtt{1}$;
- the label of the root node R_B has the lowest order;
- in any two paths from R_B to $\bot$ or $\top$, the order of the variables visited is (increasing and) the same.

Given the ordering $<$ on *Vars*, there is a unique reduced ordered BDD that represents f.

Encoding static ATs as BDDs: The semantics of an AT T can be encoded by its BDD. This is done for fault trees in [23], and the method works identically for ATs. One assumes an arbitrary order on BAS_T, and creates the reduced ordered BDD B_T of the propositional formula L_T of Sec. 3.1.

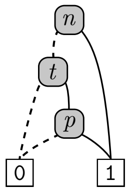
A Boolean vector $\boldsymbol{x}$ evaluates to $\mathtt{1}$ if its corresponding path in the BDD ends up in the terminal node $\top$ that is labelled $\mathtt{1}$.[3] For B_T, this means that an attack A is successful if and only if there is a path p from R_{B_T} to $\mathtt{1}$ such that the *High*-edges traversed form a subset of A. This can be phrased as follows:

Theorem 5.9 ([23]). *Let P be the set of paths $R_{B_T} \to \mathtt{1}$ in B_T. Then the map*

$$\begin{aligned} \pi\colon P &\to \mathscr{A} \\ p &\mapsto \{\mathit{Lab}(w) \in \mathit{BAS}_T \mid (w, \mathit{High}(w)) \in p\} \end{aligned}$$

satisfies $[\![T]\!] \subseteq \mathit{im}(\pi)$, and $\pi(p)$ is succesful for all $p \in P$.

Example 5.10. Let $n < t < p$ in T_s from Example 2.1: the resulting BDD (B_{T_s}) is illustrated to the right. The children of a node appear below it (so the root is on top), and a dashed line from $w \to w'$ means $w' = \mathit{Low}(w)$, and a solid line means $w' = \mathit{High}(w)$. The two paths $n \to \mathtt{1}$ correspond to the minimal attacks $\{n\}, \{t, p\}$.



An algorithm to find the (reduced ordered) BDD of an AT is given in [23]. In the worst case, the size of the BDD is exponential in the number of variables, i.e. the BASes of the AT [30]. However, DAG-structures that represent Boolean functions—such as fault trees and ATs—often have small BDD encodings [17, 42]. The choice of the linear order $<$ on BAS_T impacts the size of the BDD: finding the order that yields the smallest BDD is NP-hard, but several heuristics exists to find a good order [43, 44].

5.4 BDD-based algorithm for DAG-structured SATs

Algo. 2 (on page 9) computes metrics for DAG-structured attack trees — a similar algorithm for fault trees and their failure probability metric was introduced in [23]. Here, just like $\mathtt{BU}_{\mathtt{SAT}}$, Algo. 2 requires $D = (V, \triangledown, \vartriangle)$ to be a semiring attribute domain. However, Algo. 2 also requires the definition of neutral elements $1_\triangledown$ and $1_\vartriangle$ for $\triangledown$ and $\vartriangle$, i.e. $\forall x \in V.\ x \triangledown 1_\triangledown = x \vartriangle 1_\vartriangle = x$; we write $D_\star = (V, \triangledown, \vartriangle, 1_\triangledown, 1_\vartriangle)$ and call this a *unital semiring*. We furthermore require the domain to be absorbing — see Sec. 4.2. These conditions are mild: neutral elements are common (a semiring without them can always be extended to have them) [36]. Examples of neutral elements in Table 2 are $1_\triangledown = \infty$ and $1_\vartriangle = 0$ for min cost, and $1_\triangledown = 0$ and $1_\vartriangle = 1$ for (max) discrete probability. Moreover, most semiring metrics are absorbing, e.g. all in Table 2 are, except max challenge and max damage.

The algorithm: To explain the algorithm in more detail, we first introduce some notation. For $w \in W_n$, let $P(w)$ be the set of paths $w \to \mathtt{1}$. Furthermore, for $p \in P(w)$, define

$$\widehat{\alpha}(p) = \bigtriangleup_{w \in W_n : (w, \mathit{High}(w)) \in p} \alpha(\mathit{Lab}(w)), \tag{1}$$

$$\check{\alpha}(w) = \bigtriangledown_{p \in P(w)} \widehat{\alpha}(p). \tag{2}$$

By Theo. 5.9 each path in $P(R_{B_T})$ corresponds to a succesful attack, and $\check{\alpha}(R_{B_T})$ is calculated by performing $\triangledown$ over the metric values of these attacks. These attacks include the minimal attacks by Theo. 5.9, and because D is absorbing all other attacks are irrelevant. Hence one has $\check{\alpha}(R_{B_T}) = \check{\alpha}(T)$.

3. To ease graphical interpretation, we identify the terminal node $\top$ with its label $\mathtt{1}$ and thus speak of paths $R_{B_T} \to \mathtt{1}$.

© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: UNIVERSITA TRENTO. Downloaded on December 12,2022 at 12:12:57 UTC from IEEE Xplore. Restrictions apply.

This value can be calculated in three steps: *1.* determine $P(R_{B_T})$; *2.* for $p \in P(R_{B_T})$, calculate $\widehat{\alpha}(p)$; *3.* compute $\check{\alpha}(w)$.

Even though this approach works, it has some inefficiency built into it: there will typically be paths that share sections, and on these sections the above method calculates $\triangle$ twice. Therefore we instead use a bottom-up algorithm on the BDD, where at every node we use $\triangledown$ on the paths up to that node. The key observation is that paths $p \in P(w)$ either pass through $Low(w)$ or $High(w)$, and from that point onwards it is an element of $P(Low(w))$ or $P(High(w))$. Also, in the latter case, we need to add $\alpha(Lab(w))$ to the $\triangle$-ation when calculating $\widehat{\alpha}(p)$, because that BAS is included. By exploiting the distributivity of $\triangledown$ over $\triangle$, one can show that

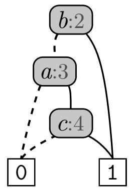
$$\check{\alpha}(w) = \check{\alpha}(Low(w)) \triangledown \big(\check{\alpha}(High(w)) \triangle \alpha(Lab(w))\big). \qquad (3)$$

In Algo. 2 we use a bottom-up method to calculate $\check{\alpha}(R_{B_T}) = \check{\alpha}(T)$ by repeatedly applying (3) from R_{B_T}. This is done until we reach the bottom nodes 0 and 1, to which we assign the values $1_\triangledown$ and $1_\triangle$, respectively.

```
Input: BDD $B_T = (W, Low, High, Lab)$,
       node $w \in W$,
       attribution $\alpha$,
       semiring attribute domain $D_\star = (V, \triangledown, \triangle, 1_\triangledown, 1_\triangle)$.
Output: Metric value $\check{\alpha}(T) \in V$.
if $Lab(w) = 0$ then
    return $1_\triangledown$
else if $Lab(w) = 1$ then
    return $1_\triangle$
else // $w \in W_n$
    return $\mathtt{BDD_{DAG}}(B_T, Low(w), \alpha, D_\star) \triangledown$
      $(\mathtt{BDD_{DAG}}(B_T, High(w), \alpha, D_\star) \triangle \alpha(Lab(w)))$
```

Algorithm 2: $\mathtt{BDD_{DAG}}$ for a DAG-structured SAT T

Example 5.11. For the DAG-structured SAT shown in Fig. 3, the order $b < a < c$ of its BASes yields the BDD to the right. Call it B_T and let us use it to compute the min cost of T like in Fig. 3, via the attribution $\alpha = \{a \mapsto 3, b \mapsto 2, c \mapsto 4\}$ and the domain $(\mathbb{N}_\infty, \min, +)$. Moreover, to use Algo. 2, we use the neutral elements $1_\triangledown = \infty$ for min and $1_\triangle = 0$ for $+$, viz. we use the attribute domain $D_\star = (\mathbb{N}_\infty, \min, +, \infty, 0)$. Let the nonterminal nodes of B_T be $W_n = \{w_a, w_b, w_c\}$, and for $w \in W$ let $\mathtt{BU}(w)$ be shorthand for $\mathtt{BDD_{DAG}}(B_T, w, \alpha, D_\star)$; this is equal to $\check{\alpha}(w)$. We compute the metric:



$$\begin{aligned}
\mathtt{BU}(R_{B_T}) &= \min\big(\mathtt{BU}(w_a),\, \mathtt{BU}(\top) + \alpha(b)\big)\\
&= \min\big(\mathtt{BU}(w_a),\, 1_\triangle + 2\big)\\
&= \min\big(\mathtt{BU}(w_a),\, 2\big)\\
&= \min\big(\min(\mathtt{BU}(\bot), \mathtt{BU}(w_c) + \alpha(a)),\, 2\big)\\
&= \min\big(\min(1_\triangledown, \mathtt{BU}(w_c) + 3),\, 2\big)\\
&= \min\big(\mathtt{BU}(w_c) + 3,\, 2\big)\\
&= \min\big(\min(\mathtt{BU}(\bot), \mathtt{BU}(\top) + \alpha(c)) + 3,\, 2\big)\\
&= \min\big(\min(1_\triangledown, 1_\triangle + 4) + 3,\, 2\big)\\
&= \min(4 + 3,\, 2) \;=\; 2.
\end{aligned}$$

To compute instead the (max) discrete probability we use the attribution $\alpha' = \{a \mapsto 0.1, b \mapsto 0.05, c \mapsto 0.6\}$ and the attribute domain $D'_\star = ([0,1]_\mathbb{Q}, \max, \cdot, 0, 1)$. Then computations are as before until the last line, which here becomes: $\max\big(\alpha'(c) \cdot \alpha'(a),\, \alpha'(b)\big) = \max(0.6 \cdot 0.1, 0.05) = 0.06$.

Theo. 5.12 states the correctness of Algo. 2, i.e. that it yields the metric for a static AT given in Def. 3.9 regardless of its structure. We prove Theo. 5.12 in Appendix D.

Theorem 5.12. *Let T be a static AT, B_T its BDD encoding over $\langle BAS, < \rangle$, α an attribution on V, and $D_\star = (V, \triangledown, \triangle, 1_\triangledown, 1_\triangle)$ an absorbing unital semiring attribute domain. Then $\check{\alpha}(T) = \mathtt{BDD_{DAG}}(B_T, R_{B_T}, \alpha, D_\star)$.*

We also note that, when actually implementing Algo. 2, one can further optimize its efficiency via dynamic programming. That is, by storing the calculated values of nodes so that this calculation is not repeated unnecessarily for nodes with multiple parents. We leave such considerations out of Algo. 2 to highlight the simple structure of the method and its relation to equation (3).

Algo. 2 has linear complexity in the size of B_T, so the overall complexity of calculating AT metrics via its BDD is mainly determined by the size of B_T. As mentioned in Sec. 5.3, this is worst-case exponential, but in practice it is usually a lot faster. This makes the BDD approach a suitable heuristic for calculating AT metrics. In Sec. 9 we show how the performance can be further improved by incorporating modular analysis.

5.5 Computing the k-top metric values

The approach described above can be extended to efficiently compute the k-top values for a given metric. This problem asks not only the min/max value of the metrics from Table 2, but also the next $k-1$ min/max values, e.g. the cost of the k cheapest attacks, or the probability of the k most likely ones.

Formally, we can express k-top values in the language of multisets. For a linearly ordered set X and a multiset $M \in \mathbb{N}^X$, we let $\min^k(M)$ be the multiset of the k smallest elements of M, with $\min^k(M) \doteq M$ when $|M| \leqslant k$. Given an attribution $\alpha\colon \text{BAS} \to X$, the *top-$k$ metric values of T* are defined as $\text{Top}_k(T, \alpha) = \min^k(\{\!\{\widehat{\alpha}(A) \mid A \in [\![T]\!]\}\!\})$ with $\{\!\{\cdot\}\!\}$ denoting a multiset. That is, $\text{Top}_k(T, \alpha)$ is an element of $\mathcal{M}(V)$, viz. a multiset of (the k smallest) values: it describes the k-top values when $\triangledown = \min$ with respect to the order on X. When $\triangledown = \max$, the k-top values are defined analogously.

The multiset $\text{Top}_k(T, \alpha)$ can be computed from the BDD using the k-shortest-paths algorithm for DAGs; this is a well-known extension of the Dijkstra (or Thorup) algorithm [45, 46]. For a DAG G with edges weighted by the matrix Q, $\mathtt{shortest\_paths}(G, Q, s, t, k, \circ)$ returns the multiset of weights of the k-shortest paths from a source node s of G, to a target node t, using operator $\circ$ to accumulate weight.

By Theo. 5.9 one has that $\text{Top}_k(T, \alpha) = \{\!\{\widehat{\alpha}(p) \mid p \in P(R_{B_T})\}\!\}$. To apply the k-shortest path algorithm, we interpret $\widehat{\alpha}(p)$ as the (weighed) length of path p, which is done by assigning weight $\alpha(Lab(w))$ to each edge $(w, High(w))$ in p, and weight $1_\triangle$ to each edge $(w, Low(w))$. We accumulate these values—to compute the length of p—with operator $\triangle$.

© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: UNIVERSITA TRENTO. Downloaded on December 12,2022 at 12:12:57 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2022.3215752

This way, the k shortest paths are the k paths with the smallest values $\widehat{\alpha}(p)$; when $\triangledown = \max$, we invert the weights.[4] This is encapsulated as Algo. 3, whose correctness is given by Theo. 5.9 and the `shortest_paths` algorithm.

```
Input: BDD $B_T = (W, Low, High, Lab)$,
       number of values to compute $k \in \mathbb{N}$,
       attribute domain $D = (V, \triangledown, \vartriangle)$,
       attribution $\alpha$.
Output: $k$-top metric values of $T$ for $\alpha$ and $D$.
$E := \bigcup_{w \in W_n} \{(w, Low(w)), (w, High(w))\}$
$Q := |W| \times |W|$ matrix filled with $1_\vartriangle$
if $\triangledown = \min$ then $sgn := 1$ else $sgn := -1$   // $\triangledown = \max$
foreach nonterminal node $w \in W_n$ do
    $Q[w][High(w)] := sgn \cdot \alpha(Lab(w))$
return $sgn \cdot$ shortest_paths$((W, E), Q, R_{B_T}, \top, k, \vartriangle)$
```

Algorithm 3: `k_top` metric values for a SAT T

Example 5.13. Consider the DAG-structured SAT from Fig. 3, $T = \text{AND}\big(\text{OR}(a, b), \text{OR}(b, c)\big)$. To compute its two cheapest attacks under the attribution $\alpha = \{a \mapsto 3, b \mapsto 1, c \mapsto 4\}$, let $b < a < c$ s.t. B_T is as in Example 5.11. The Low edge of the root b (that encodes "not performing b") is labelled with cost $1_\vartriangle = 0$, and the $High$ edge with cost $\alpha(b) = 1$; the same is done for a and c. Then the shortest-weight path from the root of B_T to its $\mathtt{1}$-labelled leaf is $p_1 = (b, \mathtt{1})$, which yields the cheapest attack $A_1 = \{b\}$ with cost $\widehat{\alpha}(A_1) = \widehat{\alpha}(p_1) = \alpha(b) = 1$. Second to that we find the path $p_2 = (b, a, c, \mathtt{1})$, which yields the second-cheapest attack $A_2 = \{a, c\}$ with cost $\widehat{\alpha}(A_2) = \widehat{\alpha}(p_2) = 1_\vartriangle \vartriangle \alpha(a) \vartriangle \alpha(c) = 0 + 3 + 4 = 7$.

In Sec. 10.2 we introduce another method to calculate $\text{Top}_k(T, \alpha)$, by expressing it as a semiring attribute domain itself. This method also extends to the dynamic case.

6 Dynamic Attack Trees

In ATs with SAND gates the execution order of the BASes becomes relevant. This affects primarily the semantics, i.e. what it means to perform a successful attack, but there are also metrics sensitive to the sequentiality of events.

6.1 Partially-ordered attacks

As for the static case, the semantics of a dynamic attack tree (DAT) is given by its successful attack scenarios. However, DATs necessitate a formal notion of order, because a sequential gate $\text{SAND}(v_1, \ldots, v_n)$ succeeds only if every v_i child is completely executed before v_{i+1} starts. This models dependencies in the order of events. For example, in Håstad's broadcast attack, n messages must first be intercepted, from which an n-th root (the secret key) may be computed. This standard interpretation is *ordered*, and an activated BAS is uninterruptedly completed. This rules out constructs that introduce circular dependencies such as $\text{SAND}(a, b, a)$.[5]

4. Technically, we assume that V has an order-inverting bijection -1. If this is not the case, then one can just invert the linear order on V when $\triangledown = \max$.

5. Cf. Kumar et al. (2015), who separate activation from execution of a BAS and can therefore operate with $\text{SAND}(a, b, a)$ [5].

Thus, an attack scenario that operates with SAND gates is not just a set $A \subseteq \text{BAS}$, but rather a partially-ordered set (poset) $\langle A, \prec \rangle$: here $\prec$ is a strict partial order, where $a \prec b$ indicates that $a \in A$ must be carried out strictly before $b \in A$. Incomparable basic attack steps can be executed in any order, or in parallel. Thus, the attack $\langle A, \prec \rangle$ indicates that all BAS in A must be executed, and their execution order will respect $\prec$. This succinct construct can represent combinatorially many execution orders of BAS. For instance $\langle \{a, b\}, \varnothing \rangle$ allows three executions: the sequence (a, b), and (b, a), and the parallel execution $a \| b$. Instead, $\langle \{a, b\}, \{(a, b)\} \rangle$ only allows the execution sequence (a, b). Note that strict partial orders are irreflexive and transitive, so e.g. $\text{SAND}(a, b, c)$ gives rise to $\prec = \{(a, b), (b, c), (a, c)\}$.

6.2 Semantics for dynamic attack trees

As for SATs we need to define the notions of attacks, suites, and the structure function. These are given below and are mostly straightforward analoga of the definitions in Sec. 3.1 in the realm of posets. The semantics as presented here were first defined in [8].

Definition 6.1. Let T be a DAT.

1) The set $\mathscr{A}_T$ of *attacks* on a DAT T is the set of strictly partially ordered sets $O = \langle A, \prec \rangle$, where $A \subseteq \text{BAS}_T$.
2) $\mathscr{A}_T$ has a partial order $\leqslant$ given by $O \leqslant O'$, for $O = \langle A, \prec \rangle$ and $O' = \langle A', \prec' \rangle$, if and only if $A \subseteq A'$ and $\prec \subseteq \prec'$.
3) An *attack suite* is a set of attacks $\mathcal{S} \subseteq \mathscr{A}_T$. The set of all attack suites is denoted $\mathscr{S}_T$.

Definition 6.2. The *structure function* $f_T \colon N \times \mathscr{A} \to \mathbb{B}$ of a dynamic AT T for $O = \langle A, \prec \rangle$ is given by

$$f_T(v, O) = \begin{cases} \mathtt{1} & \text{if } t(v) = \mathtt{BAS} \text{ and } v \in A, \\ \mathtt{1} & \text{if } t(v) = \mathtt{OR} \text{ and } \exists u \in ch(v).\, f_T(u, O) = \mathtt{1}, \\ \mathtt{1} & \text{if } t(v) = \mathtt{AND} \text{ and } \forall u \in ch(v).\, f_T(u, O) = \mathtt{1}, \\ \mathtt{1} & \text{if } t(v) = \mathtt{SAND} \text{ and } \forall u \in ch(v).\, f_T(u, O) = \mathtt{1} \\ & \quad \text{and } \forall\, 1 \leqslant i < |ch(v)|.\, \big(a \in A \cap \text{BAS}_{ch(v)_i} \wedge \\ & \qquad a' \in A \cap \text{BAS}_{ch(v)_{i+1}}\big) \Rightarrow a \prec a', \\ \mathtt{0} & \text{otherwise.} \end{cases}$$

where $\text{BAS}_v = \text{BAS} \cap ch^+(v)$ are the BAS descendants of v; and recall that $ch(v)$ is a sequence: $ch(v) = (v_1, \ldots, v_n)$.

Def. 6.2 resembles Def. 3.2 and adds SANDs: a gate $\text{SAND}(v_1, \ldots, v_n)$ succeeds iff all the BAS descendants of each v_i are completed before any BAS descendant of v_{i+1}.

Again we say that *attack O reaches v* if $f_T(v, O) = \mathtt{1}$, and write Suc_v for the suite of ("successful") attacks reaching v. Its minimal elements w.r.t. the partial order $\leqslant$ are called *minimal attacks*, and its set of minimal attacks is denoted $\llbracket v \rrbracket$. We let $\text{Suc}_T \doteq \text{Suc}_{R_T}$ and $\llbracket T \rrbracket \doteq \llbracket R_T \rrbracket$. Note that the set of minimal attacks has a different notation than we used for SATs; this is because, as we explain below, for DATs the semantics are not given by the set of minimal attacks.

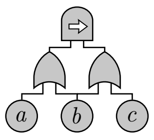
Example 6.3. Three successful attacks for the DAT T_d of Fig. 2b are: $\langle \{w, cc\}, \{(w, cc)\} \rangle$, $\langle \{ff, w\}, \varnothing \rangle$, and $\langle \{ff, w, cc\}, \{(w, cc)\} \rangle$. The first two are minimal. Attacks $\langle \{ff, cc\}, \varnothing \rangle$ and $\langle \{w, cc\}, \{(cc, w)\} \rangle$ are not successful.

© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: UNIVERSITA TRENTO. Downloaded on December 12,2022 at 12:12:57 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2022.3215752

Satisfiability of DATs: Contrary to the static case, a dynamic AT may have no successful attacks, for instance $\mathrm{Suc}_{\mathrm{SAND}(a,a)} = \varnothing$. We call T *satisfiable* if $\mathrm{Suc}_T \neq \varnothing$. Satisfiability is enforced with a notion of well-formedness in [26], where only well-formed DATs are given semantics. This enforces coherence by discarding trees that allow conflicting execution orders of BASes. However, [26] notes that this is overly restrictive, since it also discards satisfiable DATs, e.g. $\mathrm{OR}(\mathrm{SAND}(a,b), \mathrm{SAND}(b,a))$ where $\langle\{a,b\},\{(a,b)\}\rangle$ is a valid attack. Our definition of semantics avoids this issue:

Definition 6.4. The *semantics of a DAT T* is its set of successful attacks Suc_T, denoted $[\![T]\!]$.



Cf. to Def. 3.4, where the semantics of a SAT T is given by its minimal attacks $\therefore\ [\![T]\!] = \lfloor\!\lfloor T \rfloor\!\rfloor$. Instead Def. 6.4 indicates that, in general, $[\![T']\!] \supseteq \lfloor\!\lfloor T' \rfloor\!\rfloor$ for DAT T'. We use this definition because DATs are not coherent: it is possible that attack O is successful while $O' \geqslant O$ is not. For instance in the DAT of the picture to the right, $\langle\{a,c\},\{(a,c)\}\rangle$ is a successful attack, but $\langle\{a,b,c\},\{(a,c)\}\rangle$ is not: the single SAND-gate has two children that share a BAS, so not all BAS of the first child precede those of the second child.

Hence and contrary to SATs, the full semantics of a DAT cannot be recovered from just its minimal attacks. While the noncoherence of DAT semantics is a drawback compared to SAT semantics, it is needed in order to define semantics for all DAG-structured DATs, and not just the well-formed ones like in [26]. For well-formed DATs our semantics are coherent; in fact this is true for all DATs in which no two children of a SAND gate share BASes.

Note also that in spite of being laxer than [26, Def. 10], Def. 6.4 still rules out some feasible interleavings in the execution of high-level SAND gates. Consider e.g. $T' = \mathrm{SAND}\big(a, \mathrm{AND}(b,c)\big)$, where Def. 6.2 forces a to occur before any of $\{b,c\}$. However, one might argue that $\mathrm{AND}(b,c)$ is only executed once both b and c are complete, and an attack should also be considered successful as long as a is executed before $\mathrm{AND}(b,c)$. Under this interpretation (b,a,c) would be a valid execution sequence in T'. To allow this kind of sequences we need a more complex notion of strict partial order, as we discuss in Sec. 12. However, this work is about the efficient computation of metrics, and as we show next these metrics are invariant for the different valid orders of execution of BAS. Therefore, here we use the stricter but simpler semantics that stems from Def. 6.2.

Finally, Lemma 6.5 characterises the minimal attacks resulting from Def. 6.2, analogously to how Lemma 3.6 does it for static tree-structured ATs. This is key to prove the correctness of linear-time algorithms that compute metrics on tree-structured DATs. We prove this lemma in Appendix A. We note that a generalization to DAG-structured DATs, similar to Lemma 3.6, also exists, although its formulation is considerably more complicated [8]; we give the full statement in Appendix A.

Lemma 6.5. *Consider a tree-structured DAT with nodes $a \in BAS$, $v_1, v_2 \in N$. Then:*

1) $\lfloor\!\lfloor a \rfloor\!\rfloor = \{\langle\{a\},\varnothing\rangle\}$;
2) $\lfloor\!\lfloor OR(v_1,v_2) \rfloor\!\rfloor = \lfloor\!\lfloor v_1 \rfloor\!\rfloor \cup \lfloor\!\lfloor v_2 \rfloor\!\rfloor$;
3) $\lfloor\!\lfloor AND(v_1,v_2) \rfloor\!\rfloor = \{\langle A_1 \cup A_2, \prec_1 \cup \prec_2\rangle \mid \langle A_i, \prec_i\rangle \in \lfloor\!\lfloor v_i \rfloor\!\rfloor\}$;
4) $\lfloor\!\lfloor SAND(v_1,v_2) \rfloor\!\rfloor = \{\langle A_1 \cup A_2\ ,\ \prec_1 \cup \prec_2 \cup A_1 \times A_2\rangle \cdots$ $\cdots \mid \langle A_i, \prec_i\rangle \in \lfloor\!\lfloor v_i \rfloor\!\rfloor\}$;
5) *In cases 2)–4) above the $\lfloor\!\lfloor v_i \rfloor\!\rfloor$ are disjoint, and in cases 3) and 4) moreover the A_i are disjoint.*

Comparison with literature: The semantics for dynamic ATs resulting from Def. 6.4 resembles the so-called *series-parallel graphs* from [10]. We adhere to [26, 8] and define dynamic attacks as posets for a number of reasons:

- they are a succinct, natural lifting of the SAT concepts, that facilitate the extension of earlier results such as the characterisation of $\lfloor\!\lfloor\cdot\rfloor\!\rfloor$ in Lemma 6.5;
- metrics can be formally defined on this semantics, decoupling specific algorithms from a notion of correctness;
- this allows us to define algorithms to compute metrics regardless of the tree- or DAG-structure of the DAT.

The latter is different for [10], which does not work for DAG-structured DATs as noted in [18]. In the series-parallel graph semantics, BASes will occur multiple times when they have multiple parents. When calculating metrics, this leads to double counting as in Example 5.1. In contrast, posets entail a formal definition of metric over DAT semantics—given now in Sec. 6.3—which in particular yields the expected result even for DAG-structured DATs.

6.3 Security metrics for dynamic attack trees

The same fundamental concepts of metric for static ATs work for dynamic ATs: from the attributes of every BAS, obtain a metric for each attack in $\lfloor\!\lfloor T \rfloor\!\rfloor$, and from these values compute the metric for T. Thus, the generic notion of metric given by Def. 3.7 (Sec. 3.2) carries on to this section.

However, attribute domains do not suffice for DATs: metrics such as min attack time are sensitive to sequential execution of BASes. This calls for an additional *sequential operator* $\rhd\colon V^2 \to V$, to compute values of sequential parts in an attack. Therefore, metrics computation has an extra step:

0) first, an attribution α assigns a value to each BAS;
1) then, a sequential metric $\vec{\alpha}$ uses the operator $\rhd$ to assign a value to each sequential part of an attack;
2) then, a parallel metric $\widehat{\alpha}$ uses $\vartriangle$ to assign a value to each attack, as the parallel execution of its sequential parts;
3) finally, the metric $\check{\alpha}$ uses $\triangledown$ to assign a value to the whole attack suite, considering all its parallel attacks.

To formalise this operational intuition consider an attack $O = \langle A, \prec\rangle$: a *maximal chain* is a sequence $(a_1,\ldots,a_n)$ in A s.t. a_1 is minimal under $\prec$, a_n is maximal, and a_{i+1} is a direct successor of a_i for each $i < n$. The set of maximal chains in O is denoted MC_O. Thus, the 4-steps computation described above can be reinterpreted as follows:

1) $\vec{\alpha}$ uses $\rhd$ on each maximal chain, yielding one value $s_C \in V$ for each $C \in \mathrm{MC}_O$;
2) $\widehat{\alpha}$ uses $\vartriangle$ on the values $\{s_C\}_{C\in\mathrm{MC}_O}$, yielding a metric for the attack O;
3) $\check{\alpha}$ uses $\triangledown$ on the metrics of all attacks in a suite $\mathcal{S}$, yielding the metric for $\mathcal{S}$.

We use these concepts to define the metric $\check{\alpha}(\lfloor\!\lfloor T \rfloor\!\rfloor)$ of a dynamic AT T, which we denote $\check{\alpha}(T)$ in order to map Def. 6.6 to the generic notion of metric given in Def. 3.7.

© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: UNIVERSITA TRENTO. Downloaded on December 12,2022 at 12:12:57 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2022.3215752

Definition 6.6. Let $\triangledown, \vartriangle, \triangleright$ be three associative and commutative operators over a set V: we call $D = (V, \triangledown, \vartriangle, \triangleright)$ a *dynamic attribute domain*. Let T be a satisfiable dynamic AT and α an attribution on V. Let $\mathcal{S}$ be an attack suite on T. The *metric for* $\mathcal{S}$ associated to D and α is given by:

$$\check{\alpha}(\mathcal{S}) = \underbrace{\bigtriangledown_{O\in\mathcal{S}}}_{\check{\alpha}} \underbrace{\bigtriangleup_{C\in\mathrm{MC}_O}}_{\widehat{\alpha}} \underbrace{\rhd_{a\in C}}_{\vec{\alpha}} \alpha(a) \tag{4}$$

where MC_O is the set of maximal chains in the poset O. The metric for T is defined as $\check{\alpha}(T) \doteq \check{\alpha}(\llbracket T \rrbracket)$.

As for SATs, metrics are defined in terms of the minimal attacks, rather than all successful attacks. This causes a slight mismatch between the definition of metrics of DATs (based on minimal attacks) and their semantics (based on succesful attacks). We have chosen to define DAT metrics as in Def. 6.6 for consistency with SATs: in particular, when SATs are interpreted as DATs without SAND-gates, metrics such as max damage have the same value as SAT-metric and as DAT-metric. Note that when a DAT metric satisfies a suitable absorption axiom, similar to Sec. 4.2, it does not matter whether the metric is calculated from $[\![T]\!]$ or $\llbracket T \rrbracket$.

Example 6.7. The minimal attacks for the dynamic AT from Example 2.1 are $\llbracket T_d \rrbracket = \{O_1, O_2\} = \{\langle\{\mathit{ff}, w\}, \varnothing\rangle, \langle\{w, cc\}, \{w \prec cc\}\rangle\}$. The Hasse diagrams of these attacks—which resp. have one and two MCs—are shown in Figs. 4a and 4b. To compute the *min time* metric of T_d consider the attribution $\alpha = \{\mathit{ff} \mapsto 3, w \mapsto 15, cc \mapsto 1\}$ and the dynamic attribute domain $D = (\mathbb{N}, \min, \max, +)$. Then the time of the fastest attack for D and α is:

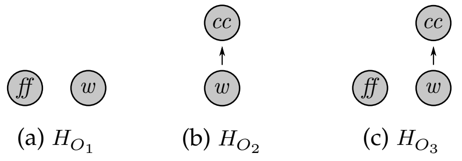
$$\begin{aligned}
\check{\alpha}(T_d) &= \bigtriangledown_{O\in\llbracket T\rrbracket} \bigtriangleup_{C\in\mathrm{MC}_O} \rhd_{a\in C} \alpha(a) \\
&= \left(\bigtriangleup_{C\in\mathrm{MC}_{O_1}} \rhd_{a\in C} \alpha(a) \right) \triangledown \left(\bigtriangleup_{C\in\mathrm{MC}_{O_2}} \rhd_{a\in C} \alpha(a) \right) \\
&= \big(\alpha(\mathit{ff}) \vartriangle \alpha(w)\big) \triangledown \big(\alpha(w) \triangleright \alpha(cc)\big) \\
&= \min(\max(3,15), 15+1) \ = \ 15.
\end{aligned}$$

Note that attack $O_1 = \langle\{\mathit{ff}, w\}, \varnothing\rangle$ has two parallel steps: two maximal chains with one node each—see Fig. 4b—so operator $\vartriangle$ has two operands with one node each: $C = \{\mathit{ff}\}$ and $C' = \{w\}$. In contrast, $O_2 = \langle\{w, cc\}, \{w \prec cc\}\rangle$ has one maximal chain with two nodes, so operator $\vartriangle$ has one operand but $\triangleright$ has two: $\alpha(w)$ and $\alpha(cc)$. Finally, the *min time* of T_d is the $\triangledown = \min$ of these two metrics: the one for O_1.

Now consider the *min skill* metric with the attributes $\alpha' = \{\mathit{ff} \mapsto 42, w \mapsto 10, cc \mapsto 0\}$. This metric is oblivious of sequential order: the skill needed to perform a task is independent of whether it must wait for the completion of other tasks. So, to compute the min skill metric of T_d we use the dynamic attribute domain $D' = (\mathbb{N}, \min, \max, \max)$, where the operators $\vartriangle$ and $\triangleright$ are the same. This results in:

$$\begin{aligned}
\check{\alpha}'(T_d) &= \big(\alpha'(\mathit{ff}) \vartriangle' \alpha'(w)\big) \triangledown' \big(\alpha'(w) \triangleright' \alpha'(cc)\big) \\
&= \min(\max(42,10), \max(10,0)) \ = \ 10.
\end{aligned}$$

Note that *the order of execution* of the BASes in the MC of an attack *is irrelevant for the value of a metric*. This is a consequence of the commutativity of the $\triangleright$ operator.



(a) H_{O_1} (b) H_{O_2} (c) H_{O_3}

Figure 4: Hasse diagrams of attacks of T_d: $O_1 = \langle\{\mathit{ff}, w\}, \varnothing\rangle$, $O_2 = \langle\{w, cc\}, \{(w, cc)\}\rangle$, $O_3 = \langle\{\mathit{ff}, w, cc\}, \{(w, cc)\}\rangle$.

As for SATs, in order to be able to actually compute metrics, we need additional structure on $(V, \triangledown, \vartriangle, \triangleright)$. More precisely, for a bottom-up algorithm to work on tree-structured DATs (see Theo. 7.1), we again need distributivity:

Definition 6.8. A *semiring dynamic attribute domain* is a dynamic attribute domain $D = (V, \triangledown, \vartriangle, \triangleright)$ where operator $\triangleright$ distributes over $\vartriangle$ and $\triangledown$, and also $\vartriangle$ distributes over $\triangledown$.

Note that min time and min skill—used in Example 6.7 above—are both semiring dynamic attribute domains.

Relation to SAT metrics: Many metrics are like min skill in Example 6.7: insensitive to the sequentiality of events. We can calculate these metrics for a DAT T by changing all SAND-gates into AND-gates, and applying our theory on (static) attribute domains. Alternatively, if $(V, \triangledown, \vartriangle)$ is a semiring attribute domain with idempotent $\vartriangle$ (such as min skill), then the metric can be calculated via Def. 6.6 for the dynamic attribute domain $(V, \triangledown, \vartriangle, \vartriangle)$.

On the other hand, any SAT T is also a DAT without SAND-gates: we disambiguate by writing T_d for the DAT interpretation. Note then that $A \in [\![T]\!]$ iff $\langle A, \varnothing\rangle \in \llbracket T_d \rrbracket$. Let then α be an attribution on a semiring attribute domain $D = (V, \vartriangle, \triangledown)$. One can extend D into a semiring *dynamic* attribute domain $(\tilde{V}, \tilde{\triangledown}, \tilde{\vartriangle}, \tilde{\triangleright})$ with $V \subseteq \tilde{V}$, s.t. $\check{\alpha}(T) = \check{\alpha}(T_d)$; for details see Appendix A. As a consequence, results proved for metrics on DATs also hold for SATs.

7 Computations for tree-structured DATs

Earlier in Example 6.7, the computation of metrics for dynamic ATs was illustrated using Def. 6.6, which is worst-case exponential in the number of nodes. However and as for SATs, there is a bottom-up algorithm to compute metrics for tree-structured DATs, that is linear in the number of nodes of the attack tree. We present a recursive version in Algo. 4 (page 13), and state its correctness in Theo. 7.1.

The proof of Theo. 7.1 (in Appendix D) relies on Theo. 9.2, whose proof in turn uses the distributivity of operator $\triangleright$ over $\triangledown$ and $\vartriangle$. Thus the fact that $(V, \triangledown, \vartriangle, \triangleright)$ is a *semiring* dynamic attribute domain is crucial.

Theorem 7.1. *Let T be a dynamic AT with tree structure, α an attribution on V, and $D = (V, \triangledown, \vartriangle, \triangleright)$ a semiring dynamic attribute domain. Then $\check{\alpha}(T) = \mathtt{BU}_{\mathtt{DAT}}(T, R_T, \alpha, D)$.*

8 Computations for DAG-structured DATs

Algo. 4 does not work for dynamic ATs with a DAG-structure, for the same reasons exposed for SATs in Sec. 5. Neither is it possible to propose algorithms based on standard BDD theory, because the computation of metrics for

© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: UNIVERSITA TRENTO. Downloaded on December 12,2022 at 12:12:57 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2022.3215752

```
Input: Dynamic attack tree $T = (N, t, ch)$,
       node $v \in N$,
       attribution $\alpha$,
       semiring dynamic attr. dom. $D = (V, \triangledown, \vartriangle, \vartriangleright)$.
Output: Metric value $\breve{\alpha}(T) \in V$.
if $t(v) =$ OR then
    return $\bigtriangledown_{u \in ch(v)}$ BU_DAT$(T, u, \alpha, D)$
else if $t(v) =$ AND then
    return $\bigtriangleup_{u \in ch(v)}$ BU_DAT$(T, u, \alpha, D)$
else if $t(v) =$ SAND then
    return $\rhd_{u \in ch(v)}$ BU_DAT$(T, u, \alpha, D)$
else // $t(v) =$ BAS
    return $\alpha(v)$
```

Algorithm 4: $\mathtt{BU}_{\mathtt{DAT}}$ for a tree-structured DAT T

DATs needs a notion of order among their BASes, that is not present in standard BDD-based data types.

As discussed in Sec. 1, some general approaches do exist to compute metrics on DAG-structured DATs [5, 9]. However, these often overshoot in terms of computation complexity. For SATs and from a procedural (rather than semantic) angle, [18] proposes a more efficient, ingenious approach that computes and then corrects a metric value by traversing the AT bottom-up repeatedly. It may be possible to extend this algorithm to cover SAND gates as well [34].

Alternatively, Def. 6.6 of DAT metric could be encoded into a naïve algorithm. This would enumerate all posets from $\llbracket T \rrbracket$, and compute the value $\breve{\alpha}(T)$ using three nested loops to traverse the corresponding Hasse diagrams. We do not expect such approach to be computationally efficient.

Instead and as in the static case, we expect that BDD encodings of the DAT offer better solutions. This requires BDD-like structures also sensitive to variable orderings. In that sense, the so-called sequential-BDDs recently presented for dynamic fault trees seem promising [47]. A first challenge would be to extend them to attributes other than failure (viz. attack) probability. Another—harder—challenge to apply this approach efficiently is the combinatorial explosion, that stems for the different possible orderings of BAS descendants of SAND gates.

In view of these considerations, we regard the algorithmic analysis for DAG-structured dynamic attack trees as an important open problem for future research. Instead, we now discuss modular analysis: a simplification strategy that can be used in any algorithm that calculates metrics.

9 Modular analysis

In this section we show that the calculation of metrics can be split up according to the modules of an AT. The resulting *modular analysis* is a well-established method for quantitative analysis of fault trees and ATs [48, 49, 50, 8]. We exploit modular analysis in the general semiring (dynamic) attribute domain setting, leading to improved performance in calculating these metrics.

For $v \in N$ we let T_v be the subDAG of T consisting of all descendants of v, with v as the root. Intuitively, a module is an inner node v such that all paths from $T \setminus T_v$ to T_v pass through v. This is formalised in the following definition.

Definition 9.1. Let $v \in N \setminus \text{BAS}$. We call node v a *module* if $T_v \cap T_w \in \{T_v, T_w, \varnothing\}$ for all $w \in N$.

Note that the root of T is always a module. The modules of an AT T can be found in linear time [48]. These modules aid calculation in the following manner: Let v be a module, then v is the only node within T_v with parents outside of T_v. This means that we can create a tree T^v by replacing T_v within T by a new single BAS $\tilde{v}$. Then the parents of $\tilde{v}$ in T^v are the parents of v in T, see Fig. 5. This allows to calculate a metric $\breve{\alpha}(T)$ by first calculating the metric on T_v, and then on T^v. This is formalized in Theo. 9.2.

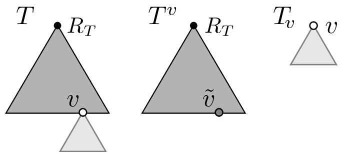


Figure 5: To compute $\breve{\alpha}(T)$ first compute the metric $\breve{\alpha}(T_v)$ for module v, then compute $\breve{\alpha}^v(T^v)$ as in Theo. 9.2.

Theorem 9.2. *Let v be a module in an AT T, and α an attribution into a (dynamic) attribute domain V. Let $T^v = (N^v, E^v)$ be the AT obtained by replacing T_v by a new single BAS $\tilde{v}$. Let $\alpha^v : N^v \to V$ be an attribution for T^v given by*

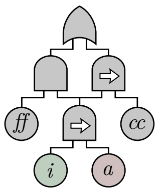
$$\alpha^v(v') = \begin{cases} \alpha(v') & \text{if } v' \neq v, \\ \breve{\alpha}(T_v) & \text{if } v' = \tilde{v}. \end{cases}$$

Then $\breve{\alpha}(T) = \breve{\alpha}^v(T^v)$.

Theo. 9.2 allows us to split up metric computation by splitting up the attack tree T. This can be used to parallelise—at least partially—any algorithm $\mathcal{A}$ that calculates a metric. More generally, when the time complexity of $\mathcal{A}$ is high, e.g. exponential in the number of nodes of T, then splitting the calculation in the modules of T will result in a lower computation time. This result is non-trivial, as it requires to express the minimal attacks of T (and their maximal chains) in terms of the minimal attacks of T^v and T_v: this is encoded in Theo. 9.2 (proved in Appendix B).

This result can be implemented by identifying all modules v of T as in [48], and then calculating $\breve{\alpha}(T_v)$ for them using any algorithm $\mathcal{A}$. By doing this bottom-up, we can use the result for lower modules in the calculation of higher ones. The resulting method is presented as Algo. 5; its correctness is stated in the following corollary of Theo. 9.2:

Corollary 9.3. *Algo. 5 correctly calculates $\breve{\alpha}(T)$.*



Example 9.4. Recall the DAT from Fig. 2b, T_d, and refine the BAS w (“walk next to victim”) to consist of two steps: “identify possible target” (i) and “approach victim” (a). Then instead of $w \in \text{BAS}$, the subtree shared by gates “skill” and “luck” becomes $v = \text{SAND}(i, a)$, yielding the DAT $T' = \text{OR}\big(\text{AND}(\mathit{ff}, v), \text{SAND}(v, \mathit{cc})\big)$. Consider the *min time* metric given by the semiring dynamic attribute domain $(\mathbb{N}, \min, \max, +)$, and attribution $\alpha = \{\mathit{ff} \mapsto 3, i \mapsto 10, a \mapsto 5, \mathit{cc} \mapsto 1\}$. Then $\breve{\alpha}(T'_v) = \alpha(i) + \alpha(a) = 15$, and since the truncated DAT T'^v is isomorphic to T_d, by Theo. 9.2 we get $\breve{\alpha}(T') = \breve{\alpha}(T_d) = 15$ (see Example 6.7). In contrast, applying eq. (4) directly to T' would have computed $\breve{\alpha}(v)$

© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: UNIVERSITA TRENTO. Downloaded on December 12,2022 at 12:12:57 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2022.3215752

```
Input: Static or dynamic attack tree $T$,
       attribution $\alpha$,
       algorithm $\mathcal{A}$ to calculate metric $\breve{\alpha}$
Output: $\breve{\alpha}(T)$.
$U := \texttt{Modules}(T)$
while $U \neq \varnothing$ do
    Extract minimal $v$ from $U$
    $\breve{\alpha}(T_v) := \mathcal{A}(T_v, \alpha)$ // Compute metric for module $v$
    $(T, \alpha) := (T^v, \alpha^v)$ // Update $\alpha$ and remove module $v$
return $\alpha(R_T)$ // $R_T$ is a BAS now
```

Algorithm 5: Modular analysis. Notation (T^v, α^v) is from Theo. 9.2; `Modules` is an algorithm that lists all modules, e.g. the one from [48].

twice. The gain of this approach is proportional to the amount of repetitions of each module, times their size.

Note that if T has tree structure, each node is a module. In this case Algo. 5 becomes a bottom-up method, and so Theo. 9.2 is an important result for Theorems 4.1 and 7.1.

Algo. 5 does not solve the problem of calculating metrics for DAG-structured DATs, since it assumes the existence of an algorithm $\mathcal{A}$ that computes $\breve{\alpha}$. However, it can speed up any found algorithm, including the enumeration of all minimal attacks to calculate the metric from these.

10 Multiple metrics simultaneously

An important class of metrics are those which assign to an AT not a single metric value, but a set of metric values. Typical examples are the following (stated below for SATs but applicable to DATs as well):

1) **Uncertainty sets**: Suppose that the $\alpha(a)$ are not known exactly, but instead we only have bounds $\alpha(a) \in [L_a, U_a]$ for all $a \in$ BAS. For instance, we might only have a confidence interval for $\alpha(a)$. In this case, we are interested in finding $\breve{L}, \breve{U}$ such that $\breve{\alpha}(T) \in [\breve{L}, \breve{U}]$.
2) ***k*-top metrics**: The k-lowest (or -highest) values of a given metric, see Sec. 5.5.
3) **Pareto front**: Attributes can be opposed, e.g. attack A_1 may be less expensive than A_2, but take more time. To understand the tradeoffs between different metrics one studies its *Pareto front*: the set of metric values of attacks that are not dominated in all metrics by another attack.

Such metrics relate to partial orders on the domain: We now give a framework to express the examples above as semiring attribute domain metrics, via their ordering. *This allows us to run once any algorithm that calculates semiring metrics, e.g. Algos. 1, 2 and 4, to compute all elements of these (multi) sets simulaneously* — proofs are in Appendix C.

Definition 10.1. 1) A *partially ordered semigroup (POSG)* is a tuple $(X, \preceq, \vartriangle)$ such that:

a) $(X, \preceq)$ is a poset;
b) $\vartriangle$ is a commutative associative operation on X;
c) If $x, y \in X$ are such that $x \preceq y$, then $x \vartriangle z \preceq y \vartriangle z$ for all $z \in X$.

If $\preceq$ is a linear order, we call $(X, \preceq, \vartriangle)$ a *linearly ordered semigroup (LOSG)*.

2) A *dynamic partially ordered semigroup (DPOSG)* is a tuple $(X, \preceq, \vartriangle, \triangleright)$ such that $(X, \preceq, \vartriangle)$ and $(X, \preceq, \triangleright)$ are POSGs and $\triangleright$ distributes over $\vartriangle$. If $\preceq$ is a linear order, we call $(X, \preceq, \vartriangle, \triangleright)$ a *dynamic linearly ordered semigroup (DLOSG)*.

There are different ways to create semiring attribute domains out of POSGs, and semiring dynamic attribute domains out of DPOSG. In the case that the partial order is linear this can be done directly:

Lemma 10.2. *1) If $(X, \preceq, \vartriangle)$ is an LOSG, then $(X, \min, \vartriangle)$ is a semiring attribute domain.*
2) If $(X, \preceq, \vartriangle, \triangleright)$ is a DLOSG, then $(X, \min, \vartriangle, \triangleright)$ is a semiring dynamic attribute domain.

Semiring attribute domains from LOSGs are ubiquitous: all examples in Table 2 come from the construction in Lemma 10.2 (reverting the natural order to change min into max if necessary). In the following we explain how examples 1)–3) mentioned above can be calculated using semiring (dynamic) attribute domains derived from (D)POSGs. We only give the dynamic statements below, but the static cases are completely analogous.

10.1 Uncertainty sets

Let $(X, \preceq, \vartriangle, \triangleright)$ be a DLOSG. Suppose that α is not known exactly; instead for every $a \in$ BAS we have $L_a, U_a \in X$ for which we know $L_a \preceq \alpha(a) \preceq U_a$. In this case, we are interested in

$$\breve{L}_T \doteq \inf\{\breve{\alpha}(T) \mid \forall a. L_a \preceq \alpha(a) \preceq U_a\}$$
$$\breve{U}_T \doteq \sup\{\breve{\alpha}(T) \mid \forall a. L_a \preceq \alpha(a) \preceq U_a\}.$$

We find this as follows: let $D = (X, \min, \vartriangle, \triangleright)$ be the semiring dynamic attribute domain from Lemma 10.2. Consider the semiring dynamic attribute domain D^2, which has underlying set X^2 and on which every operator acts componentwise. Define an attribution β with values in X^2 by $\beta(a) = (L_a, U_a)$; then $\breve{\beta}(T) = (\breve{L}_T, \breve{U}_T)$. The key observation to prove this is that for every T, the map $\alpha \mapsto \breve{\alpha}(T)$ is monotonous in each $\alpha(a)$.

10.2 *k*-top metrics

Let $(X, \preceq, \vartriangle, \triangleright)$ be a DLOSG: we want to find the k-top metric $\mathrm{Top}_k(T, \alpha) \in \mathcal{M}(X)$ from Sec. 5.5. This can be done via a semiring metric as follows. Let $\mathcal{M}^k(X)$ be the set of multisets in X of cardinality at most k. Define three operations $\triangledown^k$, $\vartriangle^k$ and $\triangleright^k$ on $\mathcal{M}^k(X)$ by

$$M_1 \triangledown^k M_2 = \min{}^k(M_1 \uplus M_2)$$
$$M_1 \vartriangle^k M_2 = \min{}^k\{\!\{x_1 \vartriangle x_2 \mid x_1 \in M_1, x_2 \in M_2\}\!\}$$
$$M_1 \triangleright^k M_2 = \min{}^k\{\!\{x_1 \triangleright x_2 \mid x_1 \in M_1, x_2 \in M_2\}\!\}.$$

Here $\uplus$ denotes multiset union. Furthermore, define a map $\beta\colon \text{BAS} \to \mathcal{M}(X)$ by $\beta(a) = \{\!\{\alpha(a)\}\!\}$. Then $\mathrm{Top}_k(T, \alpha)$ can be found as follows:

Lemma 10.3. *The tuple $(\mathcal{M}^k(X), \triangledown^k, \vartriangle^k, \triangleright^k)$ is a semiring dynamic attribute domain, and $\breve{\beta}(T) = \mathrm{Top}_k(T, \alpha)$.*

Compared to Algo. 3, this method is more general in the sense that it also works for DATs, but it comes at a complexity cost for SATs: once the BDD $(W, Low, High, Lab)$

© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: UNIVERSITA TRENTO. Downloaded on December 12,2022 at 12:12:57 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2022.3215752

Metric	Static tree	Dynamic tree	Static DAG	Dynamic DAG
min cost	BU [14, 15, 16]	BU [10]	MTBDD [17] $\mathcal{C}$-BU [18]	PTA [5]
min time	BU [14, 19]	APH [6] BU [10]	Petri nets [12]	MILP [8]
min skill	BU [14, 20]	BU [10]	$\mathcal{C}$-BU [18]	—
max damage	BU [14, 19, 20]	BU [10]	MTBDD [17] DPLL [21]	PTA [5]
probability	BU [22, 19]	APH [6]	BDD [23] DPLL [21]	I/O-IMC [9]
Pareto fronts	BU [24, 19]	**Lemma 10.4**	$\mathcal{C}$-BU [11] **Lemma 10.4**	PTA [5]
Any of the above	**Algo. 1:** $\mathtt{BU_{SAT}}$	**Algo. 4:** $\mathtt{BU_{DAT}}$	**Algo. 2:** $\mathtt{BDD_{DAG}}$	**OPEN PROBLEM**[1]
***k*-top metrics**	BU-projection [14]	BU [25] **Lemma 10.3**	**Algo. 3:** `BDD_shortest_paths`	**OPEN PROBLEM**[2]

Table 3: Algorithms for metrics on different AT classes (replica of Table 1)

BU: bottom-up on the AT structure. **APH**: acyclic phase-type (time distribution). **BDD**: binary decision diagram. **MTBDD**: multi-terminal BDD. **$\mathcal{C}$-BU**: repeated BU, identifying clones. **DPLL**: DPPL SAT-solving in the AT formula. **PTA**: priced time automata (semantics). **MILP**: mixed-integer linear programming. **I/O-IMC**: input/output interactive Markov chains (semantics). [1] Algo. 5 reduces runtime for any found method; [2] Lemma 10.3 reduces k-top calculation to metric calculation.

corresponding to T has been constructed, Algo. 3 has complexity $O(|W|+k)$, while applying Algo. 2 to Lemma 10.3 has complexity $O(k^2|W|)$.

10.3 The antichain semiring

If $(X,\preceq,\vartriangle,\triangleright)$ is a DLOSG, then we can interpret it as a semiring attribute domain $(X,\min,\vartriangle,\triangleright)$. We cannot do the same when it is a DPOSG, because $\min(x,y)$ may not exist. To create a semiring attribute domain out of a DPOSG we need a more elaborate construction. Specifically, let AC_X be the set of *antichains* in $(X,\preceq)$, i.e. sets of pairwise incomparable elements:

$$\mathrm{AC}_X = \{S \in 2^X \mid \forall x,x' \in S\colon x' \not\prec x\}.$$

Furthermore, define a map $m\colon 2^X \to \mathrm{AC}_X$ that sends a set to the antichain of its minimal elements:

$$m(S) = \{x \in S \mid \forall x' \in S\colon x' \not\prec x\}.$$

We also define three operations $\triangledown_{\mathrm{AC}}, \vartriangle_{\mathrm{AC}}, \triangleright_{\mathrm{AC}}$ on AC_X by

$$\begin{aligned} S_1 \triangledown_{\mathrm{AC}} S_2 &= m(S_1 \cup S_2)\\ S_1 \vartriangle_{\mathrm{AC}} S_2 &= m(\{x_1 \vartriangle x_2 \mid x_1 \in S_1, x_2 \in S_2\})\\ S_1 \triangleright_{\mathrm{AC}} S_2 &= m(\{x_1 \triangleright x_2 \mid x_1 \in S_1, x_2 \in S_2\}). \end{aligned}$$

The following lemma is an extension of [11], where it is shown for the static case under mild assumptions on X.

Lemma 10.4. *The tuple* $(\mathrm{AC}_X, \triangledown_{\mathrm{AC}}, \vartriangle_{\mathrm{AC}}, \triangleright_{\mathrm{AC}})$ *is a semiring dynamic attribute domain.*

This has a number of applications:

1) If $(X,\preceq,\vartriangle,\triangleright)$ is a DLOSG, then every antichain is a singleton, and the map $X \to \mathrm{AC}_X$ given by $x \mapsto \{x\}$ is an isomorphism of semiring dynamic attribute domains.
2) For a SAT T, consider the static POSG $(2^{\mathrm{BAS}}, \subseteq, \cup)$ and the attribute $\beta\colon \mathrm{BAS} \to \mathrm{AC}_{2^{\mathrm{BAS}}}$ given by $\beta(a) = \{\{a\}\}$; then $\check{\beta}(T) = [\![T]\!]$. The elements of $\mathrm{AC}_{2^{\mathrm{BAS}}}$ are suites, and for any semiring attribute domain $(V,\triangledown,\vartriangle)$, each attribution $\alpha\colon \mathrm{BAS} \to V$ induces a morphism of semiring attribute domains $\check{\alpha}\colon \mathrm{AC}_{2^{\mathrm{BAS}}} \to V$.
3) Let $(X_1,\preceq_1,\vartriangle_1,\triangleright_1),\ldots,(X_n,\preceq_n,\vartriangle_n,\triangleright_n)$ be a collection of LOSGs. Let $X = \prod_i X_i$, on which we have a partial order $\preceq$ and binary operations $\vartriangle,\triangleright$ defined componentwise. Then $(X,\preceq,\vartriangle,\triangleright)$ is a DPOSG and so $(\mathrm{AC}_X, \triangledown_{\mathrm{AC}}, \vartriangle_{\mathrm{AC}}, \triangleright_{\mathrm{AC}})$ is a semiring dynamic attribute domain. Let T be a SAT, and for each i let $\alpha_i\colon \mathrm{BAS} \to X_i$ be an attribution; let $\alpha\colon \mathrm{BAS} \to X$ be the product map. Define $\widehat{\alpha}(A) = (\widehat{\alpha}_1(A),\ldots,\widehat{\alpha}_n(A))$ for $A \in [\![T]\!]$. Then the Pareto front of T w.r.t. the α_i is the subset of X given by:

$$\mathrm{PF}(T,\alpha) = \{\widehat{\alpha}(A) \mid A \in [\![T]\!], \forall A' \in [\![T]\!]\colon \widehat{\alpha}(A') \not\prec \widehat{\alpha}(A)\}.$$

Finally, consider the map $\beta\colon \mathrm{BAS} \to \mathrm{AC}_X$ given by $\beta(a) = \{\alpha(a)\}$. Then $\check{\beta}(T) = \mathrm{PF}(T,\alpha)$.

11 Related work

Surveys on attack trees are [51, 34]: the latter covers AT analysis via formal methods, from which we are close to quantitative model checking—cf. simulation studies such as [12, 52]. Concrete case studies have been reported in [53].

Terminology in the AT literature is not uniform. In particular, some works study DAG-like structures but preserve the term "attack tree" [14, 17, 26]. Others restrict the syntactic structures to be actual trees, replicating parts of the tree—e.g. via so-called cloned nodes and repeated labels—to model the use of the same resource in several parts of an attack [13, 29, 18]. We follow the former convention, which is akin to the treatment of common cause failures in fault tree analysis [28, Sec. 8]. Thus, we write "attack tree" to refer to both tree- and DAG-like structures.

Similarly, the term *dynamic attack tree* has recently been used to refer to a set of ATs that share the main attacker's goal [54, 55]. These resemble the *attack-tree series* from [56], where "dynamic attack tree analysis" refers to the study of attack-tree series. Instead, in this work we follow [26, 27] and call an AT *dynamic* when its structure includes

© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: UNIVERSITA TRENTO. Downloaded on December 12,2022 at 12:12:57 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2022.3215752

a sequential-AND gate—so its semantics must distinguish among different execution orders of the basic attack steps. This is akin to the notions used in fault tree analysis, where dynamic gates like priority-AND in dynamic fault trees have similar semantics to sequential-AND in ATs [28, 57].

Regarding AT metrics, Table 3 condenses literature references on quantitative analyses of ATs, classified by the structure and (dynamic) gates of the ATs where they operate. For each metric and AT class, the table cites the earliest relevant contributions that include some computation procedure.

Works [22, 21] are among the first to model and compute the cost and probability of attacks: their algorithms have EXPTIME complexity regardless of the AT structure. In [5, 7] an attack is moreover characterised by the time it takes. This allows for richer Pareto analyses but introduces one clock per BAS in the Priced Time Automata semantics: algorithms have thus EXPTIME & PSPACE complexity [58, 59]. The current work improves these bounds via specialised procedures tailored for the specific AT class, e.g. Algos. 1 and 4 resp. for tree-structured SATs and DATs have LINTIME complexity.

Indeed, all algorithms specialised on tree-structured ATs implement a bottom-up traversal on its syntactic structure: we denote these BU in Table 3. Pareto analyses are polynomial, where the exponent is the number of parameters being optimised. Most works are on static ATs, with the relevant exception of [6, 10, 25] which include sequential-AND gates.

For DAG-structured static ATs the algorithmic spectrum is broader, owing to the NP-hardness of the problem (see Sec. 5.1). Such algorithms range from classical BDD encodings for probabilities, and extensions to multi-terminal BDDs, to logic-based semantics that exploit DPLL, including an encoding of SATs as generalised stochastic Petri nets. Prominent contributions are [29] and [18, Alg. 1]: after computing so-called optional and necessary clones, computations are exponential on the number of shared BAS (only). As discussed in Sec. 5.1, in this case the exponential complexity—on the number of nodes of the complete AT—lies in the input of the algorithm, i.e. clone computation. This approach is used in [11] to calculate Pareto fronts; in Lemma 10.4 we use a similar strategy but instead apply Algo. 2, whose exponential explosion lies in computing the BDD that encodes the AT.

The computation of security metrics for dynamic attack trees is more recent than for SATs: here we find open problems in the literature, indicated in two cells of Table 3. These open problems are not easy to overcome, although efficient solutions have been presented for specific cases, such as the series-parallel graph semantics of [10] which works for tree-structured DATs. However, as we discuss in Sec. 6.2 (page 11), this does not extend to DAG-structured dynamic attack trees. Another example is [9], which encodes a DAT as a (variant of a) Markov chain to compute attack probability. Min time is phrased in [8] as a mixed-integer linear programming problem. For other metrics, [5] encodes the AT as a network of PTA and solves the resulting cost-optimal reachability problem. As earlier stated, these very powerful and general approaches are in detriment of computational efficiency. Alternatively and as shown in Secs. 7 and 9, efficient (linear) bottom-up algorithms can correctly compute metrics in tree-structured DATs. This is implemented for instance in ADTool 2.0, which can also create a ranking of attacks—e.g. to find the k-top values—under the expected conditions for the operators $\triangledown, \vartriangle, \triangleright$ [25].

Regarding DAG-strutured DATs, where the open problems of Table 3 lie, recent related results encode dynamic fault trees as so-called sequential-BDDs, to compute the probability of system failure [47]. However, such safety-oriented works are hard to map to security analysis such as AT metrics because: 1) they can compute probability—and possibly parallel time—only; 2) the dynamic gates are not the same than those in dynamic ATs; 3) the standard logical gates are interpreted differently. Still, it might be feasible to adapt [47] to compute AT metrics, e.g. to compare it against the algorithms here presented. Probably the main detriment is that sequential-BDDs expand sequence dependencies of every pair of events, adding a combinatorial blow-up on top of the already exponential explosion incurred by BDD representations of DAGs. This leads us to believe that even the EXPTIME complexity of our Algo. 2 can be more efficient.

12 Conclusions

This paper presents algorithms to compute quantitative security metrics on attack trees. This is done in two steps: first, we revise and consolidate semantics in line with the literature, and we define metrics on these semantics, providing formal grounds on which to demonstrate the correctness of any devised computation method. A key contribution here is the adaptation of non-restrictive poset semantics for dynamic attack trees (Sec. 6.2), which allows for a formal definition of general metrics on a wide range of DATs.

Second, we introduce efficient and unifying algorithms that can compute many popular metrics, including sets of metrics (i.e. several metrics simultaneously as in k-top and Pareto analyses). Here, the BDD-based approach for general metrics of Algo. 2 is a prominent result, together with Lemmas 10.3 and 10.4 that show how to use single-value algorithms for the computation of set metrics.

We noted — in Sec. 6.2 — that our DAT semantics rules out some interleavings in the execution of SAND gates, e.g. (b, a, c) for $\mathrm{SAND}\big(a, \mathrm{AND}(b, c)\big)$, even when these sequences would arguably result in a succesful attack. To allow such sequences it is necessary to use formulae — rather than individual BASes— as elements of the partial order. For the DAT above, this would yield the relation $a \prec (b \wedge c)$, which allows (b, a, c) because the formulae in that sequence are satisfied in the order "first a, then $b \wedge c$." Such ordering graphs are a promising research direction.

Further lines for future work also include: developing efficient algorithms to compute metrics on DAG-structured dynamic ATs; extending our AT syntax to include *sequential-OR gates* [5, 60]; and extending our general metrics to Attack–Defense Trees [35, 61]. Other important future work is to implement the methods and algorithms from this paper in real-life case studies. Interesting future work in the opposite direction would be to frame attack tree metrics in a wider, category-theoretical framework. Operad algebras may form a useful tool for research in this direction, as attribute domains can be regarded as algebras of the operad of (dynamic) attack trees.

© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: UNIVERSITA TRENTO. Downloaded on December 12,2022 at 12:12:57 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2022.3215752

References

[1] J. Jürjens, "UMLsec: Extending UML for secure systems development," in *UML 2002 — The Unified Modeling Language*, ser. LNCS, vol. 2460, pp. 412–425. Springer Berlin Heidelberg, 2002. DOI: 10.1007/3-540-45800-X_32

[2] Y. Roudier and L. Apvrille, "SysML-Sec: A model driven approach for designing safe and secure systems," in *MODELSWARD*, pp. 655–664. IEEE, 2015. ISBN 978-989-758-136-6

[3] L. Apvrille and Y. Roudier, "SysML-sec: A sysML environment for the design and development of secure embedded systems," in *APCOSEC*, 2013. [Online]. Available: http://www.eurecom.fr/publication/4186

[4] Isograph, *AttackTree*. [Online]. Available: https://www.isograph.com/software/attacktree/

[5] R. Kumar, E. Ruijters, and M. Stoelinga, "Quantitative Attack Tree Analysis via Priced Timed Automata," in *FORTE*, ser. LNCS, vol. 9268, pp. 156–171. Springer International Publishing, 2015. DOI: 10.1007%2F978-3-319-22975-1_11

[6] F. Arnold, H. Hermanns, R. Pulungan, and M. Stoelinga, "Time-dependent analysis of attacks," in *POST*, ser. LNCS, vol. 8414, pp. 285–305. Springer Berlin Heidelberg, 2014. DOI: 10.1007/978-3-642-54792-8_16

[7] R. Kumar, S. Schivo, E. Ruijters, B. Yildiz, D. Huistra, J. Brandt, A. Rensink, and M. Stoelinga, "Effective Analysis of Attack Trees: A model-driven approach," in *FASE*, ser. LNCS, vol. 10802, pp. 56–73. Springer, 2018. DOI: 10.1007/978-3-319-89363-1_4

[8] M. Lopuhaä-Zwakenberg and M. Stoelinga, "Attack time analysis in dynamic attack trees via integer linear programming," *arXiv e-prints*, vol. abs/2111.05114, 2021. [Online]. Available: https://arxiv.org/abs/2111.05114

[9] F. Arnold, D. Guck, R. Kumar, and M. Stoelinga, "Sequential and Parallel Attack Tree Modelling," in *SAFECOMP*, ser. LNCS, vol. 9338, pp. 291–299. Springer International Publishing, 2015. DOI: 10.1007/978-3-319-24249-1_25

[10] R. Jhawar, B. Kordy, S. Mauw, S. Radomirović, and R. Trujillo-Rasua, "Attack Trees with Sequential Conjunction," in *SEC*, ser. IFIPAICT, vol. 455, pp. 339–353. Springer International Publishing, 2015. DOI: 10.1007/978-3-319-18467-8_23

[11] B. Fila and W. Wideł, "Efficient attack-defense tree analysis using Pareto attribute domains," in *CSF*, pp. 200–215, 2019. DOI: 10.1109/CSF.2019.00021

[12] Dalton, Mills, Colombi, and Raines, "Analyzing attack trees using generalized stochastic Petri nets," in *2006 IEEE Information Assurance Workshop*, pp. 116–123, 2006. DOI: 10.1109/IAW.2006.1652085

[13] M. Gribaudo, M. Iacono, and S. Marrone, "Exploiting Bayesian networks for the analysis of combined attack trees," *Electronic Notes in Theoretical Computer Science*, vol. 310, pp. 91–111, 2015. DOI: 10.1016/j.entcs.2014.12.014

[14] S. Mauw and M. Oostdijk, "Foundations of Attack Trees," in *ICISC*, ser. LNCS, vol. 3935, pp. 186–198. Springer Berlin Heidelberg, 2006. DOI: 10.1007/11734727_17

[15] J. Weiss, "A system security engineering process," in *Proceedings of the 14th National Computer Security Conference*, ser. Information System Security: Requirements & Practices, vol. 249, pp. 572–581, 1991.

[16] B. Schneier, "Attack trees," *Dr. Dobb's journal*, vol. 24, no. 12, pp. 21–29, 1999.

[17] A. Bobbio, L. Egidi, and R. Terruggia, "A methodology for qualitative/quantitative analysis of weighted attack trees," *IFAC Proceedings Volumes*, vol. 46, no. 22, pp. 133–138, 2013. DOI: 10.3182/20130904-3-UK-4041.00007

[18] B. Kordy and W. Wideł, "On quantitative analysis of attack–defense trees with repeated labels," in *POST*, ser. LNCS, vol. 10804, pp. 325–346. Springer International Publishing, 2018. DOI: 10.1007/978-3-319-89722-6_14

[19] O. Henniger, L. Apvrille, A. Fuchs, Y. Roudier, A. Ruddle, and B. Weyl, "Security requirements for automotive on-board networks," in *ITST*, pp. 641–646. IEEE, 2009. DOI: 10.1109/ITST.2009.5399279

[20] E. J. Byres, M. Franz, and D. Miller, "The use of attack trees in assessing vulnerabilities in SCADA systems," in *IISW*, pp. 3–10. IEEE, 2004.

[21] A. Jürgenson and J. Willemson, "Computing exact outcomes of multi-parameter attack trees," in *OTM*, ser. LNCS, vol. 5332, pp. 1036–1051. Springer Berlin Heidelberg, 2008. DOI: 10.1007/978-3-540-88873-4_8

[22] A. Buldas, P. Laud, J. Priisalu, M. Saarepera, and J. Willemson, "Rational choice of security measures via multi-parameter attack trees," in *CRITIS*, ser. LNCS, vol. 4347, pp. 235–248. Springer Berlin Heidelberg, 2006. DOI: 10.1007/11962977_19

[23] A. Rauzy, "New algorithms for fault trees analysis," *Reliability Engineering & System Safety*, vol. 40, no. 3, pp. 203–211, 1993. DOI: 10.1016/0951-8320(93)90060-C

[24] Z. Aslanyan and F. Nielson, "Pareto efficient solutions of attack-defence trees," in *POST*, ser. LNCS, vol. 9036, pp. 95–114. Springer Berlin Heidelberg, 2015. DOI: 10.1007/978-3-662-46666-7_6

[25] O. Gadyatskaya, R. Jhawar, P. Kordy, K. Lounis, S. Mauw, and R. Trujillo-Rasua, "Attack trees for practical security assessment: ranking of attack scenarios with ADTool 2.0," in *QEST*, ser. LNTCS, pp. 159–162. Springer, 2016. DOI: 10.1007/978-3-319-43425-4_10

[26] C. E. Budde and M. Stoelinga, "Efficient algorithms for quantitative attack tree analysis," in *CSF*, pp. 501–515. IEEE Computer Society, Jun 2021. DOI: 10.1109/CSF51468.2021.00041

[27] C. E. Budde, C. Kolb, and M. Stoelinga, "Attack trees vs. fault trees: Two sides of the same coin from different currencies," in *QEST*, ser. LNCS, vol. 12846, pp. 457–467. Springer, Aug 2021. DOI: 10.1007/978-3-030-85172-9_24

[28] W. Vesely, M. Stamatelatos, J. Dugan, J. Fragola, J. Minarick, and J. Railsback, "Fault tree handbook with aerospace applications," *NASA Office of Safety and Mission Assurance*, 2002, version 1.1.

[29] A. Bossuat and B. Kordy, "Evil twins: Handling repetitions in attack–defense trees," in *GraMSec*, ser. LNCS, vol. 10744, pp. 17–37. Springer International Publishing, 2018. DOI: 10.1007/978-3-319-74860-3_2

[30] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Transactions on Computers*, vol. C-35, no. 8, pp. 677–691, 1986. DOI: 10.1109/TC.1986.1676819

[31] W. Oortwijn, T. v. Dijk, and J. v. d. Pol, "Distributed binary decision diagrams for symbolic reachability," in *SPIN*, pp. 21–30. ACM, 2017. DOI: 10.1145/3092282.3092284

[32] Z. Aslanyan, F. Nielson, and D. Parker, "Quantitative verification and synthesis of attack-defence scenarios," in *CSF*, pp. 105–119. IEEE Computer Society, 2016. DOI: 10.1109/CSF.2016.15

[33] R. E. Barlow and F. Proschan, *Statistical theory of reliability and life testing: probability models*, ser. Intl. series in decision processes. Holt, Rinehart and Winston, 1975. ISBN 0030858534

[34] W. Wideł, M. Audinot, B. Fila, and S. Pinchinat, "Beyond 2014: Formal methods for attack tree–based security modeling," *ACM Comput. Surv.*, vol. 52, no. 4, 2019. DOI: 10.1145/3331524

[35] B. Kordy, S. Mauw, S. Radomirović, and P. Schweitzer, "Foundations of attack–defense trees," in *FAST*, ser. LNCS, vol. 6561, pp. 80–95. Springer Berlin Heidelberg, 2011. DOI: 10.1007/978-3-642-19751-2_6

[36] S. MacLane, *Categories for the working mathematician*. Springer-Verlag New York, 1971. ISBN 0387900357

[37] W. Lee, D. Grosh, F. Tillman, and C. Lie, "Fault tree analysis, methods, and applications: A review," *IEEE Transactions on Reliability*, vol. R-34, no. 3, pp. 194–203, 1985. DOI: 10.1109/TR.1985.5222114

[38] H. Hermanns and M. Siegle, "Bisimulation algorithms for stochastic process algebras and their BDD-based implementation," in *AMAST*, ser. LNCS, vol. 1601, pp. 244–264. Springer, 1999. DOI: 10.1007/3-540-48778-6_15

[39] C. Baier and J.-P. Katoen, *Principles of model checking*. MIT Press, 2008.

[40] M. Z. Kwiatkowska and D. Parker, "Advances in probabilistic model checking," in *Software Safety and Security – Tools for Analysis and Verification*, ser. NATO Science for Peace and Security Series – D: Information and Communication Security. IOS Press, 2012, vol. 33, pp. 126–151.

[41] E. Ruijters and M. Stoelinga, "Fault Tree Analysis: A survey of the state-of-the-art in modeling, analysis and tools," *Computer Science Review*, vol. 15–16, pp. 29–62, 2015. DOI: 10.1016/j.cosrev.2015.03.001

[42] A. Rauzy and Y. Dutuit, "Exact and truncated computations of prime implicants of coherent and non-coherent fault trees within Aralia," *Reliability Engineering & System Safety*, vol. 58, no. 2, pp. 127–144, 1997. DOI: 10.1016/S0951-8320(97)00034-3

[43] L. G. Valiant, "The complexity of enumeration and reliability problems," *SIAM J. Comput.*, vol. 8, no. 3, pp. 410–421, 1979. DOI: 10.1137/0208032

© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: UNIVERSITA TRENTO. Downloaded on December 12,2022 at 12:12:57 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2022.3215752

[44] M. Lê, J. Weidendorfer, and M. Walter, "A novel variable ordering heuristic for BDD-based K-terminal reliability," in *DSN*, pp. 527–537. IEEE Computer Society, 2014. DOI: 10.1109/DSN.2014.55
[45] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959. DOI: 10.1007/BF01386390
[46] M. Thorup, "Undirected single-source shortest paths with positive integer weights in linear time," *Jour. ACM*, vol. 46, no. 3, pp. 362–394, 1999. DOI: 10.1145/316542.316548
[47] H. Yu and X. Wu, "A method for transformation from dynamic fault tree to binary decision diagram," *Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability*, pp. 1–15, 2020. DOI: 10.1177/1748006X20974187
[48] Y. Dutuit and A. Rauzy, "A linear-time algorithm to find modules of fault trees," *IEEE Transactions on Reliability*, vol. 45, no. 3, pp. 422–425, 1996. DOI: 10.1109/24.537011
[49] K. A. Reay and J. D. Andrews, "A fault tree analysis strategy using binary decision diagrams," *Reliability engineering & system safety*, vol. 78, no. 1, pp. 45–56, 2002.
[50] O. Yevkin, "An improved modular approach for dynamic fault tree analysis," in *2011 Proceedings-Annual Reliability and Maintainability Symposium*, pp. 1–5. IEEE, 2011.
[51] 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. DOI: 10.1016/j.cosrev.2014.07.001
[52] Y. Wadhawan, A. AlMajali, and C. Neuman, "A comprehensive analysis of smart grid systems against cyber-physical attacks," *Electronics*, vol. 7, no. 10, 2018. DOI: 10.3390/electronics7100249
[53] M. Fraile, 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 *PoEM*, ser. LNCS, vol. 267, pp. 326–334. Springer, 2016. DOI: 10.1007/978-3-319-48393-1_24
[54] A. T. Ali and D. Gruska, "Dynamic attack trees," in *3rd Workshop on Artificial Intelligence and Formal Verification, Logic, Automata, and Synthesis*, pp. 1–5, Sep 2021.
[55] A. T. Ali and D. Gruska, "Dynamic attack trees methodology," in *IRTM*, pp. 1–9. IEEE, 2022. DOI: 10.1109/IRTM54583.2022.9791783
[56] O. Gadyatskaya and S. Mauw, "Attack-tree series: A case for dynamic attack tree analysis," in *GraMSec*, ser. LNCS, vol. 11720, pp. 7–19. Springer, 2019. DOI: 10.1007/978-3-030-36537-0_2
[57] R. E. Monti, C. E. Budde, and P. R. D'Argenio, "A compositional semantics for repairable fault trees with general distributions," in *LPAR23*, ser. EPiC Series in Computing, vol. 73, pp. 354–372. EasyChair, 2020. DOI: 10.29007/p16v
[58] R. Alur and D. L. Dill, "A theory of Timed Automata," *Theoretical Computer Science*, vol. 126, no. 2, pp. 183–235, 1994. DOI: 10.1016/0304-3975(94)90010-8
[59] G. Behrmann, K. G. Larsen, and J. I. Rasmussen, "Priced Timed Automata: Algorithms and applications," in *FMCO*, ser. LNCS, vol. 3657, pp. 162–182. Springer Berlin Heidelberg, 2005. DOI: 10.1007/11561163_8
[60] H. Hermanns, J. Krämer, J. Krčál, and M. Stoelinga, "The value of Attack-Defence Diagrams," in *POST*, ser. LNCS, vol. 9635, pp. 163–185. Springer Berlin Heidelberg, 2016. DOI: 10.1007/978-3-662-49635-0_9
[61] A. Bagnato, B. Kordy, P. H. Meland, and P. Schweitzer, "Attribute decoration of attack–defense trees," *IJSSE*, vol. 3, p. 35, 2012. DOI: 10.4018/jsse.2012040101

Milan Lopuhaä-Zwakenberg is a postdoc at University of Twente (NL), studying safety and security metrics and their interplay. Before, he was a postdoc at Eindhoven University of Technology (NL) on information-theoretic privacy metrics, and he received his PhD from Radboud University (NL) on arithmetic geometry.

Carlos E. Budde received his PhD in Computer Science in 2017 from the Universidad Nacional de Córdoba (AR), specialising in rare event simulation for formal methods. In 2017–2021 he was a postdoc at the Universiteit Twente (NL), also in collaboration with Dutch Railways. Since 2021 Carlos is assistant professor at the Università di Trento (IT), studying cybersecurity resilience via model simulation.

Mariëlle Stoelinga is a professor of risk management, both at the Radboud University and the University of Twente, in the Netherlands. Stoelinga is the project coordinator on PrimaVera, a large collaborative project on Predictive Maintenance in the Dutch National Science Agenda NWA. She also received a prestigious ERC consolidator grant. Stoelinga holds an MSc and a PhD degree from Radboud University, and has spent several years as a postdoc at the University of California at Santa Cruz, USA.

© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: UNIVERSITA TRENTO. Downloaded on December 12,2022 at 12:12:57 UTC from IEEE Xplore. Restrictions apply.