ATM: A Logic for Quantitative Security Properties on Attack Trees

Stefano M. Nicoletti[1(✉)], Milan Lopuhaä-Zwakenberg[1], Ernst Moritz Hahn[1], and Mariëlle Stoelinga[1,2]

[1] Formal Methods and Tools, University of Twente, Enschede, the Netherlands
{s.m.nicoletti,m.a.lopuhaa,e.m.hahn,m.i.a.stoelinga}@utwente.nl
[2] Department of Software Science, Radboud University, Nijmegen, The Netherlands

Abstract. Critical infrastructure systems—for which high reliability and availability are paramount—must operate securely. Attack trees (ATs) are hierarchical diagrams that offer a flexible modelling language used to assess how systems can be attacked. ATs are widely employed both in industry and academia but—in spite of their popularity—little work has been done to give practitioners instruments to formulate queries on ATs in an understandable yet powerful way. In this paper we fill this gap by presenting ATM, a logic to express quantitative security properties on ATs. ATM allows for the specification of properties involved with *security metrics* that include "cost", "probability" and "skill" and permits the formulation of insightful what-if scenarios. To showcase its potential, we apply ATM to the case study of a CubeSAT, presenting three different ways in which an attacker can compromise its availability. We showcase property specification on the corresponding attack tree and we present theory and algorithms—based on binary decision diagrams—to check properties and compute metrics of ATM-formulae.

1 Introduction

Critical infrastructure systems—for which high reliability and availability are paramount—must operate securely. Attack trees (ATs) [54] are a flexible modelling language used to assess how systems can be attacked. They operate by decomposing the attacker's goal into intermediate elements and basic attack steps that a malicious actor can take to reach said objective. ATs are widely employed both in industry and academia but—in spite of their popularity—little work has been done to give practitioners instruments to formulate queries on ATs in an understandable yet powerful way. In this paper, we fill this gap by presenting a logic to express quantitative Metrics on ATs (ATM). ATM is a powerful language able to formulate structural queries on ATs that consider quantitative security properties, or *security metrics*, such as "cost" of an attack, "probability" of getting attacked and "skill" of a malicious actor. The ability to formulate these queries is essential to provide

This work was partially funded by the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 101008233, and the ERC Consolidator Grant 864075 (*CAESAR*).

C. Ferreira and T. A. C. Willemse (Eds.): SEFM 2023, LNCS 14323, pp. 205–225, 2023.
https://doi.org/10.1007/978-3-031-47115-5_12

practitioners with an instrument to analyse what-if scenarios and to propel a more quantitatively-informed decision making process.

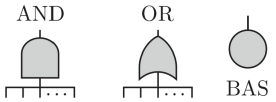
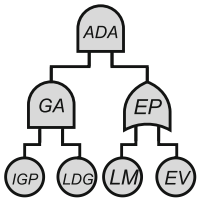


Fig. 1. Nodes in an attack tree.

Attack Trees. Attack trees (ATs) are hierarchical diagrams that represent various ways in which a system can be compromised [42,54]. Due to their popularity, ATs are referred to by many system engineering frameworks, e.g. *UMLsec* [35] and *SysMLsec* [3,53], and are supported by industrial tools such as Isograph's *AttackTree* [31]. The root—or *top level event* (TLE)—of an AT represents the attacker's goal, and the leaves represent *basic attack steps* (BASes): actions of the attacker that can no longer be refined. Intermediate nodes are labeled with gates (see Fig. 1) that determine how basic actions of the attacker can propagate to reach higher-complexity elements in the attack. ATs that do not capture dynamic behaviours present only OR and AND gates—we call these *static attack trees* (SATs)—but many extensions exist to model more elaborate attacks. To build a solid and modular foundation for our framework, this paper focuses on SATs. It is important to note that—despite their name—ATs can be *directed acyclic graphs* (DAGs), i.e., graphs in which a node may have multiple parents. Them being DAGs or tree-structured has consequences on computations [42].

Example 1. Consider the AT in Fig. 2 (excerpt from Fig. 6). This AT represents different attacks to get access to the ground station database of a CubeSAT as admin. The TLE of this sub-tree is represented by the *ADA* AND-gate. For the attacker to reach *ADA*, they have to both gain access—the *GA* AND-gate — and escalate privileges—the *EP* OR-gate. Each of these gates is then refined by BASes: to gain access, an attacker must perform information gathering and a successful phishing attack—the *IGP* BAS—and login to the ground station database using phished credentials—the *LDG* BAS. In addition, they have to *either* leverage misconfigurations—the *LM* BAS—or exploit vulnerabilities—the *EV* BAS—to escalate privileges. Note that *IGP* is represented here as a BAS but is further refined as an additional sub-tree in Fig. 6.



Access Ground Station:	
Access Ground DB as Admin	ADA
Gain Access	GA
Login to DB Ground Station	LDG
Escalate Privilege	EP
Leverage Misconfig.	LM
Exploit Vulnerabilties	EV
Info Gathering & Phishing	IGP

Fig. 2. AT modelling access to ground station DB of a CubeSAT (excerpt from Fig. 6).

Metrics on Attack Trees. ATs are often studied via *quantitative analysis*, during which they are assigned a wide range of security metrics [15,42]. Such metrics are key performance indicators that formalize how well a system performs in terms of security and are essential when comparing alternatives or making trade-offs. Typical examples of such metrics are the minimal time [5,38,39,43], minimal cost [4], or maximal probability [33] of a successful attack (see Table 1 for more examples).

1.1 Our Approach

A Logic for Attack Trees Metrics (ATM). To perform quantitatively informed decision making w.r.t. security of systems, practitioners need the ability to analyse their models in a meaningful and thorough way. As such, they must be able to formulate *meaningful queries* and meaningful *what-if scenarios*. To cater for this need, this paper presents ATM, a logic for general Metrics on Attack Trees. ATM is a flexible language used to specify properties that take metrics such as "cost", "skill" and "probability" into account directly on ATs. ATM is structured on four layers: these allow practitioners a) to reason about *successful*/*unsuccessful* attacks; b) to check whether metrics, such as the cost, are bounded by a given value on single attacks; c) to compute metrics for a class of attacks and d) to perform quantification.

Attack Trees in Practice. To offer a concrete example, we utilise ATM to specify some properties on the AT model of a CubeSAT [32,46] from the literature [22] (see Fig. 6). This model exemplifies the effect of a security threat for the availability of a system by showcasing three ways in which a malicious actor could attack a CubeSAT: performing a denial of service attack, tampering with data on the database of the ground station or killing radio communications on the satellite. Our logic can be used to specify properties on the corresponding AT and to check whether the system under examination exhibits desired characteristics. Is it necessary to leverage misconfigurations to perform a successful attack on CubeSAT's communications? Is there an attack that ensures successful access to the ground station database while keeping the cost under a certain threshold? Is there an attack that ensures data tampering without exploiting vulnerabilities in the ground station system? These are some of the properties that one could specify and check in the framework we present.

Model Checking Algorithms. In addition, we present model checking algorithms to check properties specified with ATM and to efficiently compute metrics that appear in these properties. In particular, we provide algorithms to a) check whether an AT and an attack satisfy a formula; b) compute all attacks that satisfy an AT and a formula; c) check whether the metric of a formula is bounded by a user-specified threshold; d) compute the metric value of formulae and e) check whether a quantified ATM-formula holds true. Building on previous work in the field [15,42,47,48], all these algorithms are based on construction and manipulation of binary decision diagrams (BDDs). This translation to BDDs constitutes a formal ground to address algorithmic procedures while integrating novel work presented in this paper with previously introduced frameworks.

Contributions. To summarize, in this work:

1. We develop ATM, a logic to reason about general metrics on ATs. ATM allows for the specification of metrics properties that include "cost", "proability" and "skill" and for the formulation of insightful what-if scenarios.
2. We showcase ATM by applying it to the case study of a CubeSAT and by exemplifying properties specification.
3. We propose novel algorithms based on BDDs to perform model checking and to compute metrics on properties specified using ATM.

1.2 Related Work

Numerous logics describe properties of state transition systems, such as labelled transition systems (LTSs) and Markov models, e.g., CTL [18], LTL [50], and their variants for Markov models, PCTL [28] and PLTL [49]. State-transition systems are usually not written by hand, but are the result of the semantics of high-level description mechanisms, such as AADL [12], the hardware description language VHDL [20] or model description languages such as JANI [14] or PRISM [41]. Consequently, these logics are not used to reason about the structure of such models (e.g. the placement of circuit elements in a VHDL model or the structure of modules in a PRISM model), but on the temporal behaviour of the underlying state-transition system. Similarly the majority of related work [9,11,55,56] on model checking on *fault trees* (FTs)—the safety counterpart of ATs—exhibits significant differences: these works perform model checking by referring to states in the underlying stochastic models, and properties are formulated in terms of these stochastic logics, not in terms of events in the given FT. In [57], the author provides a formulation of *Pandora*, a logic for the qualitative analysis of temporal FTs. In [27] the authors investigate how fault tree analysis (FTA) results can be linked to software safety requirements by proposing the same system model for both. They introduce a duration calculus based on discrete time interval logic (ITL) [45] to give FTs formal semantics. In [47,48] the authors present *BFL*—a logic on FTs that reasons about them in Boolean terms—and *PFL*—its probabilistic counterpart. Our work is aligned in intentions to the latter two, as we develop a logic directly on ATs. However, where they reason on FTs only in Boolean or probabilistic terms, our work exhibits a broader scope by allowing for more general queries on an ample class of security metrics. Regarding AT metrics, a seminal paper by Mauw & Oosdijk [44] shows that metrics can be computed for static ATs in a bottom-up fashion. Furthermore, [16,34] are among the first to model and compute the cost and probability of attacks. In [38,39] an attack is moreover characterised by the time it takes. In the related literature, most works are on static ATs, with the relevant exception of [5,15,25,33,42] which include sequential-AND gates. However, for static ATs the algorithmic spectrum remains broader [42]. 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 [10] and [37, Alg. 1]: after computing so-called

optional and necessary clones, computations are exponential on the number of shared BAS (only). A thorough analysis of the literature on metrics computation for ATs can be found in [15,42]. These two contributions provide efficient and general algorithms to compute security metrics on ATs. We choose to adhere to their perspective on metrics computations as it subsumes and generalizes most of the already available literature.

Structure of the Paper. Section 2 covers background on ATs, Sect. 3 presents syntax and semantics for ATM, Sect. 4 showcases an application of ATM to a CubeSAT AT, Sect. 5 presents model checking algorithms for ATM−formulae and Sect. 6 concludes the paper and discusses future work.

2 Attack Trees

Definition 1. *An* attack tree *(AT) T is a tuple (N, E, t) where (N, E) is a rooted directed acyclic graph, and $t\colon N \to \{\texttt{OR}, \texttt{AND}, \texttt{BAS}\}$ is a function such that for $v \in N$, it holds that $t(v) = \texttt{BAS}$ if and only if v is a leaf.*

Moreover, $ch\colon N \to \mathscr{P}(N)$ gives the set of *children* of a node and T has a unique root, denoted R_T. The subindex T is omitted 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 let $v = \mathrm{AND}(v_1, \ldots, v_n)$ if $t(v) = \texttt{AND}$ and $ch(v) = (v_1, \ldots, v_n)$, and analogously for OR, denoting $ch(v) = \{v_1, \ldots, v_n\}$. Furthermore, we denote the universe of ATs by $\mathscr{T}$ and call $T \in \mathscr{T}$ *tree-structured* if for any two nodes u and v none of their children is shared, else we say that T is *DAG-structured*. If only AND- and OR-gates (or their derivatives) are present we say that T is a *static attack tree* (SAT). In this paper we focus our attention on SATs and thus use the term ATs interchangeably to denote them. The semantics of a AT is defined by its successful attack scenarios, in turn given by its structure function. First, the notion of attack is defined:

Definition 2. *An* attack scenario*, or shortly an* attack*, of a static AT T is a subset of its basic attack steps: $A \subseteq BAS_T$. We denote by $\mathscr{A}_T = 2^{BAS_T}$ the universe of attacks of T. We omit the subscript when there is no confusion.*

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 adopt $\mathbb{B} = \{\texttt{1}, \texttt{0}\}$.

Definition 3. *The* structure function $f_T\colon N \times \mathscr{A} \to \mathbb{B}$ *of a static attack tree T is given by:*

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

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 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* w.r.t. T. Furthermore, the minimal attacks on R_T (i.e. the minimal successful attacks) are called *minimal attacks*. ATs are *coherent* [6], meaning that adding attack steps preserves success: if A is successful then so is $A \cup \{a\}$ for any $a \in \text{BAS}$. Thus, the suite of successful attacks of an AT is characterised by its minimal attacks.

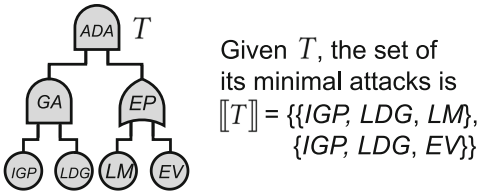


Fig. 3. All minimal attacks for AT T modelling access to ground station DB of a CubeSAT (excerpt from Fig. 6).

Definition 4. *The* semantics of an *AT* T *is its suite of minimal attacks* $[\![T]\!]$.

Example 2. Consider the AT in Fig. 3 representing ways to access the ground station database of a CubeSAT as admin: its suite of minimal attacks consists of $\{\{IGP, LDG, LM\}, \{IGP, LDG, EV\}\}$. That is, to mount a minimal attack a malicious actor needs to gain access performing information gathering and phishing *IGP*—a BAS that is further refined in Fig. 6—and by logging into the DB of the ground station; to then *either* leverage misconfigurations *LM* or exploit vulnerabilities *EV* in the DB software to gain admin privileges. A non-minimal attack on this AT would include both *LM* and *EV*.

2.1 Security Metrics for Attack Trees

Security metrics—such as the minimal time and cost among all attacks—are essential to perform quantitative analysis of systems and to support more informed decision making processes. To enable this, i.e. computing security metrics, we adopt the well-established *semiring* framework. Semirings have vast applicability potential [26] and have been successfully used to construct attribute domains on ATs [15,30,36,42]. In this paper, we formulate *linearly ordered unital semiring attribute domains* where V is the value domain, $\triangle$ is an operator to combine values of BASes in an attack, $\triangledown$ is an operator to combine values of different attacks and $\preceq$ is an order to compare values. These *linearly ordered unital semiring attribute domains* provide a convenient way to define an ample class of metrics including "min cost", "min time"—both with parallel or sequential attack steps—"min skill" and "discrete probability".

Definition 5. *A* linearly ordered unital semiring attribute domain *(simply* attribute domain *or* LOAD *from now on) is a tuple* $L = (V, \triangledown, \triangle, 1_\triangledown, 1_\triangle, \preceq)$ *where:*

- V *is a set;*
- $\triangledown, \triangle \colon V^2 \to V$ *are commutative, associative binary operations on* V*;*
- $\triangle$ *distributes over* $\triangledown$*, i.e.,* $x \triangle (y \triangledown z) = (x \triangledown y) \triangle (x \triangledown z)$ *for all* $x, y, z \in V$*;*

– ∇ *is absorbing w.r.t.* $\triangle$*, i.e.,* $x \nabla (x \triangle y) = x$ *for all* $x, y \in V$*;*
– 1_∇ *and* $1_\triangle$ *are unital elements, i.e.,* $1_\nabla \nabla x = 1_\triangle \triangle x = x$ *for all* $x \in V$*;*
– $\preceq$ *is a linear order on* V*.*

As anticipated, many relevant metrics for security analyses on ATs can be formulated as attribute domains. Table 1 shows examples, where $\mathbb{N}_\infty = \mathbb{N} \cup \{\infty\}$ includes 0 and ∞.

Example 3. An example of a LOAD is $(\mathbb{N}_\infty, \min, +, \infty, 0, \leq)$. Indeed, min and $+$ are commutative, associative operations on $\mathbb{N}_\infty$. The distributive property amounts to the fact that $x + \min(y, z) = \min(x + y, x + z)$, while the absorbing property can be stated as $\min(x, x + y) = x$. The units are given by $1_{\min} = \infty$ and $1_+ = 0$, and $\leq$ is a linear order on $\mathbb{N}_\infty$. As we will discuss in Example 4, this LOAD corresponds to the *min cost* metric on ATs.

It is important to note that derived metrics such as stochastic analyses and Pareto frontiers can be represented by semirings. However, they do not fit in this framework not being LOADs [42]. Moreover, some meaningful metrics—like the cost to defend against all attacks—do fall outside this category [42]. To render this framework functional, all BASes of ATs are enriched with attributes. More precisely, 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* $\breve{\alpha}$ assigns a value to the set of minimal attacks. We then refer to LOADs to define AT metrics. Given a LOAD $(V, \nabla, \triangle, 1_\nabla, 1_\triangle, \preceq)$ we assign to each BAS a an *attribute value* $\alpha(a) \in V$. The operators $\nabla, \triangle$ are then used to define a metric value for T as follows:

Table 1. AT metrics with attribute domains.

Metric	V	∇	$\triangle$	1_∇	$1_\triangle$	$\preceq$
min cost	$\mathbb{N}_\infty$	min	$+$	∞	0	$\leq$
min time (sequential)	$\mathbb{N}_\infty$	min	$+$	∞	0	$\leq$
min time (parallel)	$\mathbb{N}_\infty$	min	max	∞	0	$\leq$
min skill	$\mathbb{N}_\infty$	min	max	∞	0	$\leq$
discrete prob	$[0, 1]$	max	$\cdot$	0	1	$\leq$

Definition 6. *Let* T *be an AT and let* $L = (V, \nabla, \triangle, 1_\nabla, 1_\triangle, \preceq)$ *be a LOAD.*

1. *An* attribution *on* T with values in L *is a map* $\alpha \colon BAS_T \to V$*;*
2. *Given such* α*, define the* metric value *of an attack* A *by*

$$\widehat{\alpha}(A) = \bigtriangleup_{a \in A} \alpha(a);$$

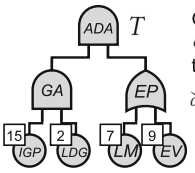
3. *Given such* α*, define the* metric value *of* T *by*

$$\breve{\alpha}(T) = \bigtriangledown_{A \in [\![T]\!]} \widehat{\alpha}(A) = \bigtriangledown_{A \in [\![T]\!]} \bigtriangleup_{a \in A} \alpha(a).$$

Example 4. Consider L from Example 3 representing the metric *min cost*, and let T be the AT in Fig. 4. To each BAS we attach a cost value, given by the attribution $\alpha : \mathrm{BAS}_T \to V$ given by $\{IGP \mapsto 15, LDG \mapsto 2, LM \mapsto 7, EV \mapsto 9\}$.

As in Example 2, T has two minimal attacks, $A_1 = \{IGP, LDG, LM\}$ and $A_2 = \{IGP, LDG, EV\}$. Since $\triangle = +$, We have $\widehat{\alpha}(A_1) = \alpha(IGP) + \alpha(LDG) + \alpha(LM) = 15 + 2 + 7 = 24$; this is the cost an attacker needs to spend to perform attack A_1. Similarly one finds $\widehat{\alpha}(A_2) = 15 + 2 + 9 = 26$. We then calculate $\breve{\alpha}(T) = \min(\widehat{\alpha}(A_1), \widehat{\alpha}(A_2)) = 24$. Indeed, the minimal cost incurred by an attacker to succesfully attack the system is by performing the cheapest minimal attack, which is A_1.

When computing multiple metrics on a given AT, one can resort to multiple LOADs and coherently chosen attributions over its BASes. We thus define such a tree as follows:



Given T and an attribution over BASes
$\alpha : \{IGP \mapsto 15, LDG \mapsto 2, LM \mapsto 7, EV \mapsto 9\}$,
then *min cost* for T is calculated as follows:

$$\begin{aligned}\breve{\alpha}(T) &= \widehat{\alpha}(\{IGP, LDG, LM\}) \triangledown \widehat{\alpha}(\{IGP, LDG, EV\}) \\ &= \widehat{\alpha}(\{IGP \triangle LDG \triangle LM\}) \triangledown \widehat{\alpha}(\{IGP \triangle LDG \triangle EV\}) \\ &= \min(15 + 2 + 7, 15 + 2 + 9) = \min(24, 26) = 24.\end{aligned}$$

Fig. 4. Computing *min cost* for T: AT for accessing a ground station DB of a CubeSAT (excerpt from Fig. 6).

Definition 7. *An* attributed AT *is a tuple* $\mathsf{T} = (T, \mathscr{L}, \mathfrak{a})$ *where: 1.* T *is an attack tree; 2.* $\mathscr{L} = \{L_1, \ldots, L_l\}$ *is a set of LOADs; 3.* $\mathfrak{a} = \{\alpha_i\}_{i=1}^{l}$ *is a set of attributions on* T*, where each* α_i *takes values in* L_i*.*

Although in this paper we calculate metrics by considering all *minimal* attacks—coherently with [15,42]—one could also simply consider all successful attacks. For metrics obtained from LOADs 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.

3 A Logic for at General Metrics

3.1 Syntax of ATM

Below, we present ATM, a logic for general Metrics on Attack Trees. ATM shares the objective of developing a language directly on tree-shaped models with [47, 48]. However, it extends the scope of these works to the security domain and allows for property specification that consider a large class of security metrics. The syntax of ATM is structured on four layers. The first layer, ϕ, reasons about the status of elements in an AT. Atomic formulae e represent BASes and IEs in an AT and they can be combined with usual Boolean connectives. Furthermore, we can forcefully set the value of an element in a layer 1 formula to either 0 or 1 with $\phi[e \mapsto 0]$ and $\phi[e \mapsto 1]$. With $\mathrm{MA}(\phi)$ we can check whether an attack is a *minimal attack*, i.e., a minimal attack successful for a given ϕ. Layer two and three reason about metrics. Layer 2 formulae allow the user to check whether a given metric on a ϕ formula is bounded by m ($\mathbb{M}_k(\phi) \preceq_k m$) and to forcefully

set the attribution of a given $e \in \psi$ to an appropriate value ν ($\psi[e \overset{k}{\mapsto} \nu]$). Boolean connectives are also allowed. Layer 3 formulae also allow the setting of attributions but simply return the *value* of a calculated metric ($\mathbb{V}_k(\phi)$). Note that for the layer 1, layer 2 and layer 3 formulae we usually assign values with $\mapsto$ to $e \in \texttt{BAS}$. We can however assign values to IEs if 1. e is a module [21], i.e., all paths between descendants of e and the rest of the AT pass through e 2. and none of the descendants of e are present in the formula. If so, we prune that (sub-)AT and treat occurring IEs as BASes. Finally, layer 4 formulae allow us to perform quantification over layer 1 and layer 2 formulae. Given a set of LOADs $\mathscr{L} = \{L_1, \ldots, L_l\}$ with $L_k \in \mathscr{L}$ and $m \in V_k$ the syntax is defined as follows:

$$\text{Layer 1: } \phi ::= e \mid \neg\phi \mid \phi \wedge \phi \mid \phi[e \mapsto 0] \mid \phi[e \mapsto 1] \mid \text{MA}(\phi)$$

$$\text{Layer 2: } \psi ::= \neg\psi \mid \psi \wedge \psi \mid \mathbb{M}_k(\phi) \preceq_k m \mid \psi[e \overset{k}{\mapsto} \nu]$$

$$\text{Layer 3: } \xi ::= \mathbb{V}_k(\phi) \mid \xi[e \overset{k}{\mapsto} \nu]$$

$$\text{Layer 4: } \gamma ::= \neg\gamma \mid \exists(\phi \wedge \psi) \mid \forall(\phi \wedge \psi)$$

Syntactic Sugar. We define the following derived operators, where formulae θ are either layer 1 or layer 2 formulae.

$$\theta_1 \vee \theta_2 ::= \neg(\neg\theta_1 \wedge \neg\theta_2) \qquad \theta_1 \not\Leftrightarrow \theta_2 ::= \neg(\theta_1 \Leftrightarrow \theta_2)$$
$$\theta_1 \Rightarrow \theta_2 ::= \neg(\theta_1 \wedge \neg\theta_2) \qquad \text{MD}(\phi) ::= \text{MA}(\neg\phi)$$
$$\theta_1 \Leftrightarrow \theta_2 ::= (\theta_1 \Rightarrow \theta_2) \wedge (\theta_2 \Rightarrow \theta_1)$$

where $\text{MD}(\phi)$ checks whether A is a *minimal defence* w.r.t. ϕ, i.e., a set that guarantees that ϕ is not reached.

3.2 Semantics of ATM

The semantics for our logic reflect objects needed to evaluate the four syntactical layers. For the first layer of ATM, formulae are evaluated on an attack A and on a tree T. Atomic formulae e are satisfied by A and T if the structure function in Definition 3 returns 1 with A and e as input. Formally:

$$\begin{array}{ll}
A, T \models e & \text{iff } f_T(e, A) = 1 \\
A, T \models \neg\phi & \text{iff } A, T \not\models \phi \\
A, T \models \phi \wedge \phi' & \text{iff } A, T \models \phi \text{ and } A, T \models \phi' \\
A, T \models \phi[e_i \mapsto 0] & \text{iff } A', T \models \phi \text{ with } A' = \{a'_1, \ldots, a'_n\} \text{ where} \\
 & a'_i = 0 \text{ and } a'_j = a_j \text{ for } j \neq i \\
A, T \models \phi[e_i \mapsto 1] & \text{iff } A', T \models \phi \text{ with } A' = \{a'_1, \ldots, a'_n\} \text{ where} \\
 & a'_i = 1 \text{ and } a'_j = a_j \text{ for } j \neq i \\
A, T \models \text{MA}(\phi) & \text{iff } A \in [\![\phi]\!]_T
\end{array}$$

With $[\![\phi]\!]_T$ we denote the *minimal satisfaction set* of attacks for ϕ, i.e., the set of minimal attacks that satisfy ϕ given T. We define $[\![\phi]\!]_T$ as follows: $[\![\phi]\!]_T = \{A \mid$

$A, T \models \phi \wedge \nexists A' \subseteq A.A', T \models \phi\}$. It is important to note that—with semantics defined as we did—we allow for fairly granular reasoning over ATs. In particular, we can evaluate whether an attack compromises a particular sub-AT without *reaching* the TLE. Semantics for the second and third layer require *attributed trees* (see Definition 7). We can then define semantics for the second layer:

$$\begin{array}{ll} A, \mathsf{T} \models \neg\psi & \text{iff } A, \mathsf{T} \not\models \psi \\ A, \mathsf{T} \models \psi \wedge \psi' & \text{iff } A, \mathsf{T} \models \psi \text{ and } A, \mathsf{T} \models \psi' \\ A, \mathsf{T} \models \mathbb{M}_k(\phi) \preceq_k m & \text{iff } A, T \models \phi \wedge \widehat{\alpha}\,(A) \preceq_k m \\ A, \mathsf{T} \models \psi[e_i \overset{k}{\mapsto} \nu] & \text{iff } A, \mathsf{T}(\mathfrak{a}[\alpha_k(a_i) \overset{k}{\mapsto} \nu]) \models \psi \end{array}$$

For an attack A and an attributed tree T to satisfy $\mathbb{M}_k(\phi) \preceq_k m$, both the attack A and the tree T must satisfy the inner layer 1 formula and the security metric calculated on the attack must respect the given threshold. We let X_1 be the set of layer 1 formulae and we define a *ϕ-security metric* to attribute a value to a layer 1 formula:

Definition 8. *A* ϕ-security metric *is a function* $\breve{\alpha}^{\mathsf{T}} \colon X_1 \to V$ *defined as follows:*

$$\breve{\alpha}^{\mathsf{T}}(\phi) \;=\; \bigtriangledown_{A \in [\![\phi]\!]_T} \;\; \bigtriangleup_{a \in A} \; \alpha(a).$$

Note that in Definition 8 some occurrences can lead to the application of the $\breve{\alpha}$ function to the empty set, i.e., when $[\![\phi]\!]_T = \varnothing$. To account for this, we resort to $1_\bigtriangledown$ and $1_\bigtriangleup$ for $\bigtriangledown$ and $\bigtriangleup$ (see Definition 5). Assuming the case in which $\breve{\alpha}^{\mathsf{T}}(\phi) \equiv \breve{\alpha}(\varnothing)$, we fix that $\breve{\alpha}^{\mathsf{T}}(\phi) = 1_\bigtriangledown$; likewise for $\widehat{\alpha}$ and $1_\bigtriangleup$. Furthermore, with $\breve{\alpha}^{\mathsf{T}_k} \colon X_1 \to V_k$ we denote a ϕ-security metric whose domain and attribution are obtained appropriately from the k-est LOAD $L_k \in \mathscr{L}$. We then let $\mathfrak{a}[\alpha_k(a_i) \overset{k}{\mapsto} \nu]$ be the attribution on the element $a_i \in A$ via α_k to an arbitrary value ν, chosen appropriately from the domain V_k of L_k. Consequently, we define semantics for the third layer. Let $\mathsf{Val}_{\mathsf{T}} \colon X_3 \to V_k$ define an evaluation function of layer three formulae in X_3:

$$\begin{array}{ll} \mathsf{Val}_{\mathsf{T}}(\mathbb{V}_k(\phi)) & = \breve{\alpha}^{\mathsf{T}_k}(\phi) \\ \mathsf{Val}_{\mathsf{T}}(\xi[e_i \overset{k}{\mapsto} \nu]) & = \mathsf{Val}_{\mathsf{T}(\mathfrak{a}[\alpha_k(a_i) \overset{k}{\mapsto} \nu])}(\xi) \end{array}$$

Finally, we can define semantics for the fourth layer containing quantifiers:

$$\begin{array}{ll} \mathsf{T} \models \neg\gamma & \text{iff } \mathsf{T} \not\models \gamma \\ \mathsf{T} \models \exists(\phi \wedge \psi) & \text{iff } \exists A \,.\, A, T \models \phi \text{ and } A, \mathsf{T} \models \psi \\ \mathsf{T} \models \forall(\phi \wedge \psi) & \text{iff } \forall A \,.\, A, T \models \phi \text{ and } A, \mathsf{T} \models \psi \end{array}$$

4 Case Study: Attacking a CubeSAT

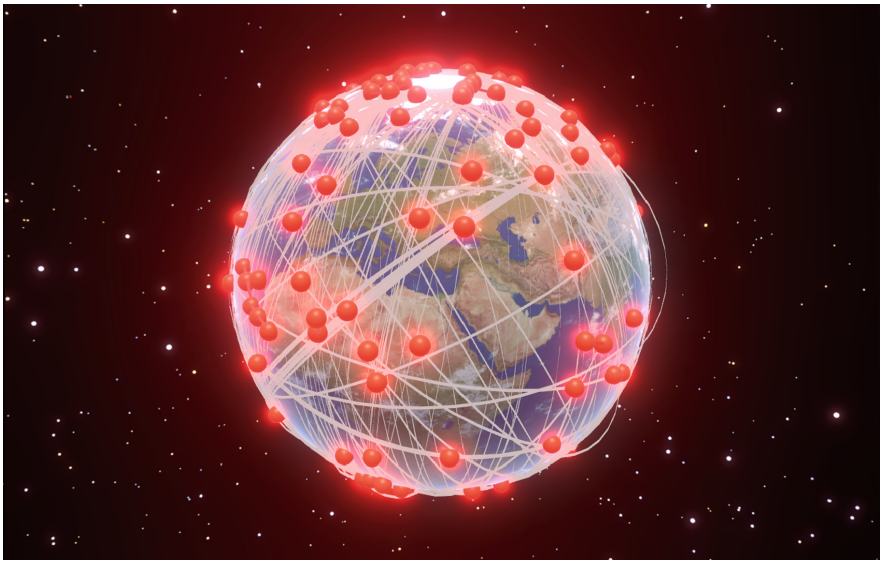


Fig. 5. Representation of orbiting CubeSATs.

CubeSATs are a type of *nanosatellite* typically used for academic and educational purposes [46]: they are usually built in units (or "U") of 10cm x 10cm x 10cm and can be combined to form larger satellites. They are relatively inexpensive to design, build, and launch compared to traditional, larger satellites and they are a popular choice among students, universities, technology pioneers, and crowd-sourced initiatives [32]. To give a sense of the importance of CubeSATs in our orbital ecosystem, we provide a representation of orbiting CubeSATs as of March 2023 in Fig. 5 and an animation in [23]. A total of 153 elements are plotted on the Earth, following data provided by the online database Celestrack [17]. The size of each sphere is exaggerated for visual purposes—a diameter of 500km for each element—and satellites are propagated using the Simplified General Perturbations 4 (SGP4) orbit propagator [29]. As CubeSATs are one of the platforms achieving more consensus in the context of the "New Space" [19,32], it is fundamental that security risks on these systems are not overlooked. To cater for this need, we showcase how ATM can be applied to specify useful properties on CubeSATs.

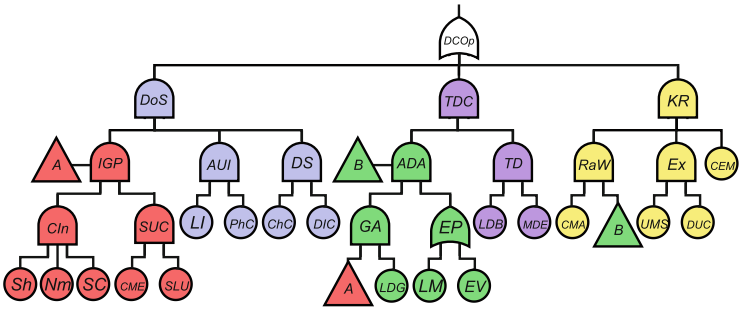


Fig. 6. An AT representing ways to attack a CubeSAT.

Table 2. Abbreviations for the AT in Fig. 6.

CubeSAT TLE:		Info Gathering + Phish:		Data Tampering:	
Disrupt CubeSAT Operations	DCOp	Info Gathering & Phishing	IGP	Tamper Data from CubeSAT	TDC
DoS Attack:		Collect Information	CIn	Tamper with Data	TD
Denial of Service	DoS	Shodan	Sh	Login to DB as Admin	LDB
Access CubeSAT UI	AUI	NMAP	Nm	Modify Database Entries	MDE
Locate Interfaces	LI	Scrape Credentials	SC	**Kill Comms on CubeSAT:**	
Login with Phished Creds	PhC	**Access Ground Station:**		Kill Radio on CubeSAT	KR
Disrupt Service	DS	Access Ground DB as Admin	ADA	Recon. and Weaponization	RaW
Change Config. Settings	ChC	Gain Access	GA	Create Malicious App	CMA
Delete Items on CubeSAT	DIC	Login to DB Ground Station	LDG	Exploit	Ex
Steal User Credentials	SUC	Escalate Privilege	EP	Upload Malware to Server	UMS
Craft Malicious Email	CME	Leverage Misconfig.	LM	Command for Upload	EV
Send as Legit User	SLU	Exploit Vulnerabilties	EV	SAT Gets & Exec. Malware	CEM

In Fig. 6, an AT represents three possible ways in which an attacker could compromise the availability of a CubeSAT. The scenario and the original ATs are taken from [22] and then slightly adapted to model a unique cohesive AT. The TLE in Fig. 6 represents the disruption of CubeSAT's operations—the *DCOp* OR-gate. This gate is detailed by three children: *DoS*—the indigo TLE of a sub-tree on the left presenting a denial of service attack—*TDC*—the violet TLE of the central sub-tree detailing a data tampering attack—and *KR*—the yellow TLE of the sub-tree on the right that presents an attack killing communications on the CubeSAT. For a denial of service to happen, the attacker must perform information gathering and a successful phishing attack—detailed by the red *IGP* AND-gate—and use gathered intel to access the CubeSAT UI and disrupt the service. On the other hand, to perform a data tampering attack, one must access the ground control database as admin—detailed by the green *ADA* AND-gate—then modify database entries and tamper with data. Finally, to kill communications on the CubeSAT an attacker must perform reconnaissance and weaponization, crafting a malicious app, and also conduct the exploit uploading the malware on the CubeSAT via the ground station: executing this code on the satellite would cause communications to go offline. Due to the increasing

complexity of these three different attacks, the AT in Fig. 6 presents several sub-trees that are shared. The red sub-tree for information gathering and phishing is shared by the denial of service attack and by the sub-tree that models getting access to the database on the ground station. Furthermore, this green sub-tree is itself shared between the tampering data attack and the more complex malware-based communication killing attack.

Properties. ATM allows us to specify some properties on the AT in Fig. 6. As per semantics, properties 2 and 3 are evaluated w.r.t. a given attack. 1) What are all minimal attacks to achieve denial of service? $[\![DoS]\!]_T$; 2) Are the cost of data-tampering and info gathering and phishing respectively lower than 20 and at most 5? $\mathsf{Cost}(TDC) < 20 \wedge \mathsf{Cost}(IGP) \leq 5$; 3) Are the probability of successfully attacking the TLE and the parallel time of attack lower than 0.05 and 45 respectively? $\mathsf{Prob}(DCOp) < 0.05 \wedge \mathsf{ParTime}(DCOp) < 45$; 4) What is the min skill an attacker has to have to kill communications on the CubeSAT, assuming that one needs skill of 20 to perform info gathering and phishing? $\mathsf{Skill}(KR)[IGP \mapsto 20]$; 5) Is there an attack that ensures data tampering without exploiting vulnerabilities in the ground station system? $\exists(TDC[EV \mapsto 0])$; 6) Is it necessary to leverage misconfigurations to perform a successful attack on CubeSAT's communications? $\forall(KR \Rightarrow LM)$; 7) Is there an attack that ensures successful access to the ground station DB while keeping the cost under 20? $\exists\,(\mathsf{Cost}(ADA) < 20)$; 8) Do accessing the CubeSAT UI and disrupting service always imply that successful attacks to the TLE are strictly cheaper than 35 and strictly faster than 60 (when parallelized)? $\forall((AUI \wedge DS) \Rightarrow (\mathsf{Cost}(DCOp) < 35 \wedge \mathsf{ParTime}(DCOp) < 60))$ (Table 2).

5 Model Checking Algoritms

In this section we present model checking algorithms for ATM. As noted in [47, 48], some scenarios, especially in the Boolean domain, are trivial: e.g., checking if $A, T \models \phi$ holds is trivial if ϕ is a formula that does not contain a MA or MD operator. In that case, we can simply substitute the values of A in the atoms of ϕ and see if the Boolean expression evaluates to true. Non trivial scenarios arise if ϕ contains a MA or MD operator or if ATs are not tree-shaped. These require computations based on BDDs, introduced in Sect. 5.1: a coherent choice with the landscape of algorithms for FT logics [47,48] and AT computation [15,42]. In this section we build upon these results and present algorithms to: 1) Obtain BDDs from layer 1 formulae taking the structure of a given tree T into account (Sect. 5.2); 2) a) Check whether an attack A and a tree T satisfy a layer 1 formula and b) compute all the satisfying attacks A for a given tree T and layer 1 formula (Sect. 5.3); 3) Check whether an attack A and an attributed tree T satisfy a layer 2 formula (Sect. 5.4); 4) Compute the metric value of a given layer 3 formula (Sect. 5.5); 5) Check whether an attributed tree T satisfies a layer 4 formula (Sect. 5.6).

5.1 Binary Decision Diagrams (BDDs)

BDDs are directed acyclic graphs (DAGs) that compactly represent Boolean functions [2] by reducing redundancy. Depending on variable's ordering, BDD's size can grow linearly in the number of variables and at worst exponentially. In practice, BDDs are heavily used, including in AT analysis [15,42] and in their safety counterpart, FTs [7,47,48,52]. Formally, a BDD is a rooted DAG B_f that represents a Boolean function $f\colon \mathbb{B}^n \to \mathbb{B}$ over variables $Vars = \{x_i\}_{i=1}^n$. Each nonleaf w has two outgoing arrows, labeled $\mathtt{0}$ and $\mathtt{1}$, and a label $Lab(w) \in Vars$; furthermore, each leaf has a label $\mathtt{0}$ or $\mathtt{1}$. Given a b in $\mathbb{B}^n$, the BDD is used to compute $f(b)$ as follows: starting from the top, upon arriving at a node w with $Lab(w) = x_i$, one takes the $\mathtt{0}$-edge if $b_i = 0$ and the $\mathtt{1}$-edge if $b_i = 1$. The label of the leaf one ends up in, is then equal to $f(b)$. A function f can be represented by multiple BDDs, but has a unique *reduced ordered* representative, or ROBDD [8,13], where the x_i occur in ascending order, and the BDD is reduced as much as possible by removing irrelevant nodes and merging duplicates. This is formally defined below; we let $Low(w)$ (resp. $High(w)$) be the endpoint of w's $\mathtt{0}$-edge (resp. $\mathtt{1}$-edge) and let R_B be the BDD root.

Definition 9. *Let Vars be a set. A* (RO)BDD *over Vars is a tuple* $B = (W, H, Lab, u)$ *where* (W, H) *is a rooted directed acyclic graph, and* $Lab\colon W \to Vars \sqcup \{\mathtt{0}, \mathtt{1}\}, u\colon H \to \{\mathtt{0}, \mathtt{1}\}$ *are maps such that: 1. Every nonleaf w has exactly two outgoing edges* h, h' *with* $u(h) \neq u(h')$*, and* $Lab(w) \in Vars$*; 2. Every leaf w has* $Lab(w) \in \{\mathtt{0}, \mathtt{1}\}$*. 3. Vars are equipped with a total order,* B_f *is thus defined over a pair* $\langle Vars, < \rangle$*; 4. the variable of a node is of lower order than its children, that is:* $\forall\, w \in W_n.\, Lab(w) < Lab(Low(w))$, $Lab(High(w))$*; 5. the children of nonleaf nodes are distinct nodes; 6. nodes are uniquely determined by their label, low child and high child.*

5.2 BDDs from ATs and Layer 1 Formulae

The first step to enable further computations is to obtain BDDs from layer 1 formulae taking the structure of a given tree T into account (for related procedures on FT logics see [47,48]). Following, operations between BDDs are represented by **bold** operands e.g., $\boldsymbol{\wedge}$, $\boldsymbol{\vee}$. Where convenient notationally, we write B_T^ϕ for $B_T(\phi)$, i.e., the BDD B of ϕ, given T. Given a set of variables $Vars = \{x_i\}_{i=1}^n$ existential quantification can be defined as follows: $\boldsymbol{\exists}\, x.\, B = \textsc{Restrict}(B, x, 0) \boldsymbol{\vee} \textsc{Restrict}(B, x, 1)$ and $\boldsymbol{\exists} Vars.\, B = \boldsymbol{\exists} x_1 . \boldsymbol{\exists} x_2 . \ldots \boldsymbol{\exists} x_n .\, B$. Furthermore, we define a set of primed variables $Vars' = \{x_i'\}_{i=1}^n$ and let $B_T^\phi[Vars \curvearrowright Vars']$ be the BDD B_T^ϕ in which every variable $x_i \in Vars$ is renamed to its primed $x_i' \in Vars'$. Finally we let $Vars' \subset Vars \equiv (\bigwedge_i x_i' \Rightarrow x_i) \wedge (\bigvee_i x_i' \neq x_i)$. Algorithms to conduct typical BDD operations—such as Restrict—can be found in [2,8].

Definition 10. *The translation function of an AT T is a function* $\mathfrak{f}_T\colon \mathtt{E} \to \mathtt{BDD}$ *that takes as input an element* $e \in \mathtt{E}$.

$$\mathfrak{f}_T(e) = \begin{cases} B(e) & \text{if } e \in \mathtt{BAS} \\ \bigvee_{e' \in ch(e)} \mathfrak{f}_T(e') & \text{if } e \in \mathtt{IE} \text{ and } t(e) = \mathtt{OR} \\ \bigwedge_{e' \in ch(e)} \mathfrak{f}_T(e') & \text{if } e \in \mathtt{IE} \text{ and } t(e) = \mathtt{AND} \end{cases}$$

where $B(e)$ *is a BDD with a single node* w *with* $Low(w) = \mathtt{0}$ *and* $High(w) = \mathtt{1}$.

Algorithm 1 computes B_T^ϕ following semantics for layer 1 formulae:

Algorithm 1. Compute B_T^ϕ from T and ϕ

Input: AT T, formula ϕ
Output: BDD B_T^ϕ
Method:
if $\phi = e$ **then return** $\mathfrak{f}_T(e)$
else if $\phi = \neg\phi'$ **then return** $\neg\,(Algorithm\,1(T, \phi'))$
else if $\phi = \phi' \wedge \phi''$ **then return** $Algorithm\,1(T, \phi') \wedge Algorithm1(T, \phi'')$
else if $\phi = \phi'[e_i \mapsto 0]$ **then return** $\textsc{Restrict}(Algorithm\,1(T, \phi'), x_i, 0)$
else if $\phi = \phi'[e_i \mapsto 1]$ **then return** $\textsc{Restrict}(Algorithm\,1(T, \phi'), x_i, 1)$
else // $\phi = \mathrm{MA}(\phi')$
    return $Algorithm\,1(T, \phi') \wedge (\neg\exists\, Vars'.(Vars' \subset Vars) \wedge$
    $Algorithm\,1(T, \phi')[Vars \rightsquigarrow Vars'])$
end if

5.3 Model Checking Layer 1 Formulae

Is an Attack Successful w.r.t. ϕ? Algorithm 2 checks whether $A, T \models \phi$, given an attack A and a tree T. First, the BDD B_T^ϕ for ϕ given T is constructed via Algorithm 1. Then, the algorithm walks the BDD path representing values of BASes in A. If it ends up in the terminal $\mathtt{0}$, then $A, T \not\models \phi$, otherwise—if the terminal node is $\mathtt{1}$—$A, T \models \phi$.

Algorithm 2. Check if $A, T \models \phi$

Input: attack A, attack tree T, formula ϕ
Output: *true* iff $A, T \models \phi$; *false* otherwise.
Method:
$B_T^\phi \leftarrow Algorithm\,1(T, \phi)$; $w_i = R_{B_T^\phi}$
while $w_i \notin W_t$ **do**:
    if $a_i \in A = 0$ **then** $w_i = Low(w_i)$
    else if $a_i \in A = 1$ **then** $w_i = High(w_i)$
    end if
end while

10: **if** $Lab(w_i) = 0$ **then return** *false*
11: **else** // $Lab(w_i) = 1$
12: **return** *true*
13: **end if**

All Successful Attacks w.r.t. ϕ. Our ability to construct a BDD B_T^ϕ for layer 1 formulae granted by Algorithm 1 allows us to compute all attacks A such that $A, T \models \phi$. Algorithm 3 performs this computation by applying the AllSat [2] algorithm to B_T^ϕ: AllSat walks down the BDD and stores the paths that lead to the terminal node 1. These paths then represent satisfying attacks for ϕ given T. Note that Algorithm 3 can be used to compute all the minimal attacks of a given ϕ by simply calling it on MA(ϕ).

5.4 Model Checking Layer 2 Formulae

Algorithm 4 presented in this subsection checks if a layer 2 formula is satisfied, given an attack A and an attributed tree T. Boolean connectives are resolved as usual via case distinction. To check whether $A, \mathsf{T} \models \mathbb{M}_k(\phi) \preceq_k m$, first the BDD B_T^ϕ for the inner layer 1 formula is constructed and Algorithm 2 is emplyed to assess whether $A, T \models \phi$. If that is not the case, we return *false*. Otherwise, we compute the metric value for the given attack following the interpretation of $\triangle$ taken from the $k-$est LOSG L_k of our attributed tree T. We store this value in `metr_val`, and we return the result of the comparison with $\preceq_k m$. To handle the case in which we set evidence for a specific atom e_i in a layer 2 formula, we simply call the algorithm again and we make sure that the attribution α_k of the corresponding a_i is mapped to the chosen value ν.

Algorithm 3. Compute all A s.t. $A, T \models \phi$

1: **Input**: AT T, formula ϕ
2: **Output:** $\{A \mid A, T \models \phi\}$
3: **Method:**
4: $B_T^\phi \leftarrow$ *Algorithm* 1(T, ϕ); $w_i = R_{B_T^\phi}$
5: $\{A \mid A, T \models \phi\} \leftarrow$ AllSat(w_i)
6: **return** $\{A \mid A, T \models \phi\}$

Algorithm 4. Check if $A, \mathsf{T} \models \psi$

1: **Input**: attack A, attributed AT T, formula ψ
2: **Output:** *true* iff $A, \mathsf{T} \models \psi$; *false* otherwise.
3: **Method:**
4: **if** $\psi = \neg\psi'$ **then return not** *Algorithm* 4(A, T, ψ')
5: **else if** $\psi = \psi' \wedge \psi''$ **then return** *Algorithm* 4(A, T, ψ') **and** *Algorithm* 4(A, T, ψ'')
6: **else if** $\psi = \mathbb{M}_k(\phi) \preceq_k m$ **then**
7: **if** *Algorithm* 2(A, T, ϕ) returns *true* **then** // $A, T \models \phi$
8: `metr_val` $= \bigwedge_{a \in A}^{k} \alpha_k(a)$
9: **return** `metr_val` $\preceq_k m$

10: **else** // $A, T \not\models \phi$
11: **return** *false*
12: **end if**
13: **else** // $\psi = \psi'[e_i \overset{k}{\mapsto} \nu]$
14: **return** *Algorithm* 4$(A, \mathsf{T}(\mathfrak{a}[\alpha_k(a_i) \overset{k}{\mapsto} \nu]), \psi')$
15: **end if**

5.5 Compute Metrics for Layer 3 Formulae

This subsection showcases an algorithm to compute a metric value for a specified ξ-formula. If ξ equals $\mathbb{V}_k(\phi)$, one approach would be to directly use the formula of Definition 8. However, directly finding all minimal attacks on ϕ is computationally expensive [42]. Instead, we calculate metrics by applying the BDD-based method from [42]. This method exploits the fact that paths from the root to 1 in a BDD encode succesful attacks, and 1-labeled edges on such a path represent the BAS of these attacks. Assigning weight $\alpha(Lab(w))$ to an edge $(w, High(w))$, the metric value can then be computed by a variant of the shortest path algorithm for DAGs. Note that the method in [42] is defined only for $\phi = e$, but the result readily generalizes. If $\xi = \xi'[e_i \overset{k}{\mapsto} \nu]$, the algorithm is called again on ξ' and the attribution α_k on the corresponding a_i is set to ν.

Algorithm 5. Compute metric for ξ-formula

1: **Input**: attributed ATT, formula ξ
2: **Output:** metric value for ξ.
3: **Method:**
4: **if** $\xi = \mathbb{V}_k(\phi)$ **then**
5: $(W, H, Lab, u) \leftarrow$ *Algorithm* 1(T, ϕ)
6: $W_{\text{todo}} \leftarrow W$
7: **while** $W_{\text{todo}} \neq \varnothing$ **do**
8: Take $w \in W_{\text{todo}}$ without children in W_{todo}
9: **if** $Lab(w) = \mathtt{0}$ **then** $v(w) \leftarrow 1_\triangledown$
10: **else if** $Lab(w) = \mathtt{1}$ **then** $v(w) \leftarrow 1_\triangle$
11: **else**
12: $v(w) \leftarrow v(Low(w)) \triangledown (v(High(w)) \triangle \alpha(Lab(w)))$
13: **end if**
14: $W_{\text{todo}} \leftarrow W_{\text{todo}} \setminus \{w\}$
15: **end while return** $v(\mathrm{R}_{W,H,Lab,u})$
16: **else** // $\xi = \xi'[e_i \overset{k}{\mapsto} \nu]$
17: **return** *Algorithm* 5$(\mathsf{T}(\mathfrak{a}[\alpha_k(a_i) \overset{k}{\mapsto} \nu]), \xi')$
18: **end if**

5.6 Model Checking Layer 4 Formulae

We present an algorithm to check whether an attributed tree T satisfies a layer 4 formula. The non-trivial cases of Algorithm 6 check whether $\mathsf{T} \models \exists(\phi \wedge \psi)$ and $\mathsf{T} \models \forall(\phi \wedge \psi)$. In the former case, for each attack A_i in the set of satisfying attacks for ϕ — $\{A \mid A, T \models \phi\} \leftarrow$ *Algorithm* 3(T, ϕ)—we check whether $A_i, \mathsf{T} \models \psi$. If we find a fitting A_i, we return it alongside *true*. Otherwise, we return *false*. In the latter case, for each A_i in the set of all attacks for T $\mathscr{A}_T$ we check whether either $A_i, T \not\models \phi$ *or* $A_i, \mathsf{T} \not\models \psi$. If we find a counterexample A_i, we return it alongside *false*. Otherwise, we return *true*.

Algorithm 6. Check if $\mathsf{T} \models \gamma$

Input: set of all attacks $\mathscr{A}_T$, attributed ATT, formula γ
Output: *true* iff $\mathsf{T} \models \gamma$; *false* otherwise; (counter)example A_i.
Method:
if $\gamma = \neg\gamma'$ **then return not** *Algorithm* 6(A, T, γ')
else if $\gamma = \exists(\phi \wedge \psi)$ **then**
  for $A_i \in \{A \mid A, T \models \phi\} \leftarrow$ *Algorithm* 3(T, ϕ) **do**
    if *Algorithm* 4(A_i, T, ψ) returns *true* **then return** *true*, A_i
    end if
  end for
  return *false*
else if $\gamma = \forall(\phi \wedge \psi)$ **then**
  for $A_i \in \mathscr{A}_T$ **do**
    if *Algorithm* 2(A_i, T, ϕ) returns *false* $\vee$ *Algorithm* 4(A_i, T, ψ) returns *false* **then**
      return *false*, A_i
    end if
  end for
  return *true*
end if

6 Conclusions

We presented ATM, a logic for general metrics on ATs that enables the construction of complex queries and insightful what-if scenarios. We showcased its usefulness with an application of ATM to the case study of a CubeSAT. Specified properties can then be checked and metrics computed via model checking algorithms that we presented. Our work opens several relevant perspectives for future research. First, it would be interesting to extend ATM to consider timed behaviours: this would allow to further extend quantitative analysis capabilities. This step could be achieved by extending ATM to dynamic ATs that consider the sequential nature of attack steps. To handle dynamic gates in dynamic ATs it would be very natural to have a logic that can express temporal properties, moving more in the direction of LTL [50] or CTL [18] or their timed variants TLTL [51] and TCTL [1]. Another notable extension of ATM could express and calculate Pareto fronts between metrics [42]. Moreover, it is foreseeable to extend the proposed framework to safety-security variants of ATs and FTs, e.g., to attack-fault trees (AFTs) [40], and to graphs that consider more general safety-security *risks*, in the sense of *probability* $\times$ *impact* [24]. Lastly, implementing this logic could further propel usability of ATM by providing hands-on feedback from domain experts acquainted with threat modelling and vulnerability analysis.

Acknowledgements. The authors would like to thank Dr. Juan A. Fraire(0000-0001-9816-6989) (Inria, CONICET and Saarland University) for the insightful discussions about routing in space and for propagating and visualizing orbiting CubeSATs, resulting in Fig. 5 and in the animation in [23].

References

1. Alur, R., Courcoubetis, C., Dill, D.: Model-checking in dense real-time. Inf. Comput. **104**(1), 2–34 (1993)
2. Andersen, H.R.: An intro. to binary decision diagrams. Lecture notes, available online, IT University of Copenhagen, p. 5 (1997)
3. Apvrille, L., Roudier, Y.: SysML-sec: a sysML environment for the design and development of secure embedded systems. In: APCOSEC (2013)
4. Arnold, F., Guck, D., Kumar, R., Stoelinga, M.: Sequential and parallel attack tree modelling. In: Koornneef, F., van Gulijk, C. (eds.) SAFECOMP 2015. LNCS, vol. 9338, pp. 291–299. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-24249-1_25
5. Arnold, F., Hermanns, H., Pulungan, R., Stoelinga, M.: Time-dependent analysis of attacks. In: Abadi, M., Kremer, S. (eds.) POST 2014. LNCS, vol. 8414, pp. 285–305. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-54792-8_16
6. Barlow, R.E., Proschan, F.: Statistical theory of reliability and life testing: probability models. In: International Series in Decision Processes, Holt, Rinehart and Winston (1975)
7. Basgöze, D., Volk, M., Katoen, J., Khan, S., Stoelinga, M.: BDDs strike back - efficient analysis of static and dynamic fault trees. In: NFM, vol. 13260, pp. 713–732 (2022)
8. Ben-Ari, M.: Mathematical Logic for Computer Science. Springer, Heidelberg (2012). https://doi.org/10.1007/978-1-4471-4129-7
9. Bieber, P., Castel, C., Seguin, C.: Combination of fault tree analysis and model checking for safety assessment of complex system. In: EDCC, vol. 2485, pp. 19–31 (2002)
10. Bossuat, A., Kordy, B.: Evil twins: handling repetitions in attack–defense trees. In: Liu, P., Mauw, S., Stølen, K. (eds.) GraMSec 2017. LNCS, vol. 10744, pp. 17–37. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-74860-3_2
11. Boudali, H., Crouzen, P., Stoelinga, M.: Dynamic fault tree analysis using input/output interactive markov chains. In: DSN, pp. 708–717 (2007)
12. Bozzano, M., Cimatti, A., Katoen, J., Nguyen, V.Y., Noll, T., Roveri, M.: Safety, dependability and performance analysis of extended AADL models. Comput. J. **54**(5), 754–775 (2011)
13. Brace, K.S., Rudell, R.L., Bryant, R.E.: Efficient implementation of a BDD package. In: 27th ACM/IEEE Design Automation Conference, pp. 40–45 (1990)
14. Budde, C.E., Dehnert, C., Hahn, E.M., Hartmanns, A., Junges, S., Turrini, A.: JANI: quantitative model and tool interaction. In: Legay, A., Margaria, T. (eds.) TACAS 2017. LNCS, vol. 10206, pp. 151–168. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-54580-5_9
15. Budde, C.E., Stoelinga, M.: Efficient algorithms for quantitative attack tree analysis. In: CSF, pp. 1–15 (2021)
16. Buldas, A., Laud, P., Priisalu, J., Saarepera, M., Willemson, J.: Rational choice of security measures via multi-parameter attack trees. In: Lopez, J. (ed.) CRITIS 2006. LNCS, vol. 4347, pp. 235–248. Springer, Heidelberg (2006). https://doi.org/10.1007/11962977_19
17. Celestrack: Orbiting CubeSATs (2023). https://celestrak.org/NORAD/elements/gp.php?GROUP=cubesat&FORMAT=tle. Accessed Mar 2023

18. Clarke, E.M., Emerson, E.: Design and synthesis of synchronisation skeletons using branching time temporal logic. In: Logic of Programs, Proceedings of Workshop, LNCS, vol. 31, pp. 52–71 (1981). Springer, Heidelberg. https://doi.org/10.1007/bfb0025774
19. CORDIS, European Commission: MISSION (2023). https://cordis.europa.eu/project/id/101008233
20. Déharbe, D., Shankar, S., Clarke, E.M.: Model checking VHDL with CV. In: Gopalakrishnan, G., Windley, P. (eds.) FMCAD 1998. LNCS, vol. 1522, pp. 508–514. Springer, Heidelberg (1998). https://doi.org/10.1007/3-540-49519-3_33
21. Dutuit, Y., Rauzy, A.: A linear-time algorithm to find modules of fault trees. IEEE Trans. Reliab. **45**(3), 422–425 (1996)
22. Falco, G., Viswanathan, A., Santangelo, A.: Cubesat security attack tree analysis. In: SMC-IT, pp. 68–76 (2021)
23. Fraire, J.: All active CubeSATs as of 2023 (according to Celestrak). https://www.youtube.com/watch?v=PIkwxOvPLTw. Accessed Aug 2023
24. Fumagalli, M., et al.: On the semantics of risk propagation. In: International Conference on Research Challenges in Information Science, pp. 69–86. Springer, Heidelberg (2023). https://doi.org/10.1007/978-3-031-33080-3_5
25. Gadyatskaya, O., Jhawar, R., Kordy, P., Lounis, K., Mauw, S., Trujillo-Rasua, R.: Attack trees for practical security assessment: ranking of attack scenarios with ADTool 2.0. In: Agha, G., Van Houdt, B. (eds.) QEST 2016. LNCS, vol. 9826, pp. 159–162. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-43425-4_10
26. Golan, J.S.: Semirings and their Applications. Springer, Heidelberg (2013). https://doi.org/10.1007/978-94-015-9333-5
27. Hansen, K.M., Ravn, A.P., Stavridou, V.: From safety analysis to software requirements. IEEE Trans. Softw. Eng. **24**(7), 573–584 (1998)
28. Hansson, H., Jonsson, B.: A logic for reasoning about time and reliability. Formal Aspects Comput. **6**(5), 512–535 (1994)
29. Hejduk, M.D., Casali, S.J., Cappellucci, D.A., Ericson, N.L., Snow, D.: A catalogue-wide implementation of general perturbations orbit determination extrapolated from higher order orbital theory solutions. In: Proceedings of the 23rd AAS/AIAA Space Flight Mechanics Meeting, pp. 619–632 (2013)
30. Horne, R., Mauw, S., Tiu, A.: Semantics for specialising attack trees based on linear logic. Fund. Inf. **153**(1–2), 57–86 (2017)
31. Isograph: AttackTree. https://www.isograph.com/software/attacktree/. Accessed Mar 2023
32. Jet Propulsion Laboratory NASA: CubeSATs and SmallSATs. https://www.jpl.nasa.gov/topics/cubesats. Accessed Mar 2023
33. Jhawar, R., Kordy, B., Mauw, S., Radomirović, S., Trujillo-Rasua, R.: Attack trees with sequential conjunction. In: Federrath, H., Gollmann, D. (eds.) SEC 2015. IAICT, vol. 455, pp. 339–353. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-18467-8_23
34. Jürgenson, A., Willemson, J.: Computing exact outcomes of multi-parameter attack trees. In: Meersman, R., Tari, Z. (eds.) OTM 2008. LNCS, vol. 5332, pp. 1036–1051. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-88873-4_8
35. Jürjens, J.: UMLsec: extending UML for secure systems development. In: UML 2002 – The Unified Modeling Language, vol. 2460, pp. 412–425 (2002)
36. Kordy, B., Pouly, M., Schweitzer, P.: Probabilistic reasoning with graphical security models. Inf. Sci. **342**, 111–131 (2016)

37. Kordy, B., Wideł, W.: On quantitative analysis of attack–defense trees with repeated labels. In: Bauer, L., Küsters, R. (eds.) POST 2018. LNCS, vol. 10804, pp. 325–346. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-89722-6_14
38. Kumar, R., Ruijters, E., Stoelinga, M.: Quantitative attack tree analysis via priced timed automata. In: Sankaranarayanan, S., Vicario, E. (eds.) FORMATS 2015. LNCS, vol. 9268, pp. 156–171. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-22975-1_11
39. Kumar, R., et al.: Effective analysis of attack trees: a model-driven approach. In: Russo, A., Schürr, A. (eds.) FASE 2018. LNCS, vol. 10802, pp. 56–73. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-89363-1_4
40. Kumar, R., Stoelinga, M.: Quantitative security and safety analysis with attack-fault trees. In: HASE, pp. 25–32 (2017)
41. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 585–591. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22110-1_47
42. Lopuhaä-Zwakenberg, M., Budde, C.E., Stoelinga, M.: Efficient and generic algorithms for quantitative attack tree analysis. IEEE TDSC **20**, 4169–4187 (2022)
43. Lopuhaä-Zwakenberg, M., Stoelinga, M.: Attack time analysis in dynamic attack trees via integer linear programming. arXiv e-prints arXiv:2111.05114 (2021)
44. Mauw, S., Oostdijk, M.: Foundations of attack trees. In: Won, D.H., Kim, S. (eds.) ICISC 2005. LNCS, vol. 3935, pp. 186–198. Springer, Heidelberg (2006). https://doi.org/10.1007/11734727_17
45. Moszkowski, B.: A temporal logic for multi-level reasoning about hardware. STANFORD UNIV CA, Technical report (1982)
46. NASA: CubeSATs Overview. https://www.nasa.gov/mission_pages/cubesats/overview. Accessed Mar 2023
47. Nicoletti, S., Hahn, E., Stoelinga, M.: BFL: a logic to reason about fault trees. In: DSN, pp. 441–452 (2022)
48. Nicoletti, S.M., Lopuhaä-Zwakenberg, M., Hahn, E.M., Stoelinga, M.: Pfl: a probabilistic logic for fault trees. In: FM 2023, pp. 199–221 (2023)
49. Ognjanovic, Z.: Discrete linear-time probabilistic logics: completeness, decidability and complexity. J. Log. Comput. **16**(2), 257–285 (2006)
50. Pnueli, A.: The temporal logic of programs. In: FOCS, pp. 46–57 (1977)
51. Raskin, J.F.: Logics, automata and classical theories for deciding real time. Ph.D. thesis (1999)
52. Rauzy, A.: New algorithms for fault trees analysis. RESS **40**(3), 203–211 (1993)
53. Roudier, Y., Apvrille, L.: SysML-Sec: a model driven approach for designing safe and secure systems. In: MODELSWARD, pp. 655–664. IEEE (2015)
54. Schneier, B.: Attack trees. Dr. Dobb's J. **24**(12), 21–29 (1999)
55. Thums, A., Schellhorn, G.: Model checking FTA. In: FME, vol. 2805, pp. 739–757 (2003)
56. Volk, M., Junges, S., Katoen, J.: Fast dynamic fault tree analysis by model checking techniques. Trans. Ind. Inf. **14**(1), 370–379 (2018)
57. Walker, M.D.: Pandora: a logic for the qualitative analysis of temporal fault trees. Ph.D. thesis, The University of Hull (2009)