Parametric Analyses of Attack-fault Trees

Étienne André, Didier Lime, Mathias Ramparison, Mariëlle Stoelinga

Parametric analyses of attack-fault trees

Étienne André[1,2,3], Didier Lime[4], Mathias Ramparison[5,6,7], Mariëlle Stoelinga[8]

[1]Université de Lorraine, CNRS, Inria, LORIA, Nancy, France
[2]JFLI, CNRS, Tokyo, Japan
[3]National Institute of Informatics, Tokyo, Japan
[4]École Centrale de Nantes, LS2N, CNRS, UMR 6004, Nantes, France
[5]Université Sorbonne Paris Nord, LIPN, CNRS, F-93430, Villetaneuse, France
[6]University of Luxembourg, Luxembourg
[7]Univ. Grenoble Alpes, CNRS, Grenoble INP, VERIMAG, 38000 Grenoble, France
[8]Formal Methods and Tools, University of Twente, The Netherlands

Abstract

Risk assessment of cyber-physical systems, such as power plants, connected devices and IT-infrastructures has always been challenging: safety (i. e., absence of unintentional failures) and security (i. e., no disruptions due to attackers) are conditions that must be guaranteed. One of the traditional tools used to consider these problems is attack trees, a tree-based formalism inspired by fault trees, a well-known formalism used in safety engineering. In this paper we define and implement the translation of attack-fault trees (AFTs) to a new extension of timed automata, called parametric weighted timed automata. This allows us to parameterize constants such as time and discrete costs in an AFT and then, using the model-checker IMITATOR, to compute the set of parameter values such that a successful attack is possible. Moreover, we add the possibility to define counter-measures. Using the different sets of parameter values computed, different attack and fault scenarios can be deduced depending on the budget, time or computation power of the attacker, providing helpful data to select the most efficient counter-measure.

Keywords— security, attack-fault trees, parametric timed automata, IMITATOR

1 Introduction

In the past few years, the range of security breaches in the security of organizations has become larger and larger. The process of unifying them by determining relations

This manuscript is the author version of the manuscript of the same name published in Fundamenta Informatica 182(1). The final authenticated version is available at https://www.doi.org/10.3233/FI-2021-2066. This manuscript is an extended version of the manuscript of the same name [And+19] published in the proceedings of the 19th International Conference on Application of Concurrency to System Design (ACSD 2019). This work is partially supported by the ANR national research program PACS (ANR-14-CE28-0002), the ANR-NRF French-Singaporean research program ProMiS (ANR-19-CE25-0015), the PHC Van Gogh project PAMPAS, by STW under the project 15474 SEQUOIA, KIA KIEM project 628.010.006 StepUp, the EU under the project 102112 SUCCESS and ERATO HASUO Metamathematics for Systems Design Project (No. JPMJER1603), JST.

between and consequences of separated events has become more difficult: how to link the presence of solid oxygen in a helium tank with the explosion of SpaceX rocket Falcon 9 during firing tests? What is the cost for the attacker and damages caused to SpaceX manufacturing plants? One of the tools available to help structure risk assessments and security analyses is *attack trees*, recommended, e. g., by NATO Research and Technology Organisation (RTO) [RR08] and OWASP (Open Web Application Security Project) [13]. *Attack trees* [Sal+98] were formalized in [Kor+10] as a popular and convenient formalism for security analysis (see [KPS14] for a survey) and are inspired by *fault trees* [FMC09; RS15], a well-known formalism used in safety engineering. Bottom-up computation for a single parameter (e. g., cost, probability or time of an attack), can be performed directly on attack trees [Bag+12]. Attack trees and fault trees are quite similar but differ on their gates and/or goals [Bag+12; Kor+14]. Both are constructed with leaves that model component and attack step failures or successes that propagate through the system via gates. While fault trees focus on safety properties, attack trees considerate skills, resources and risk appetite possessed by an attacker performing actions. *Attack-fault trees* (AFTs) [KS17] combine safety properties from fault trees and security conditions from attack trees; therefore gates of both fault trees and attack trees are used in this formalism.

Quantitative analysis of AFTs with multiple quantitative annotations on AFTs like cost, time, failure probabilities—which can functionally be dependent on each other—evaluates risks and helps to figure out the most risky scenarios and therefore to select the most effective counter-measures.

Contribution In this work, we study a more abstract version of the security problem, and we propose an approach to *synthesize* times and costs necessary to individual actions in order to perform a successful attack or individual failures causing the failure of the entire system. The global attack time and cost can then be expressed as a combination of the parametric unit costs. To this end, we propose a formalization of attack-fault trees using an *ad-hoc* extension of parametric timed automata called *parametric weighted timed automata* (PWTAs). PWTAs can be seen as a generalization of parametric timed automata (PTAs) [AHV93] and weighted/priced automata [Beh+01; ALP04] with only costs on transitions.

We implement our framework within the tool ATTop presented in [Kum+18], allowing to define AFTs in the Galileo format, and provide an automated translation into the IMITATOR input format [And21].

As a proof of concept, we apply our framework to an attack tree of [KS17] and an original attack-fault tree. With the help of the parametric timed model checker IMITATOR, we are able to synthesize constraints in several dimensions; further we discuss induced possible attack and fault scenarios.

This enlarges the scope of quantitative analysis for AFTs by parameterizing multiple annotations on the AFT *at once* such as time, cost and damages and then compute for instance the optimal combination of parameter values for the attack to fail quickly while keeping damages to the system low.

We further extend our framework to attack-defense trees [Kor+14].

Related work Risk assessment is a wide research area, studied over the past years through different methods [ODK99; WSJ07] and standards [MSR06]. More specifically, the formalism of attack trees has been studied through lattice theory [Kor+10], laying the formal foundations of the modeling methodology. Attack tree analysis later

required more powerful analysis techniques to deal with complex behaviors of systems. Analysis through Bayesian networks [GIM15] is a first step in the definition of a method to reuse and combine attack trees. Transformations of attack trees into input-output interactive Markov chains and continuous time Markov decision process have been studied in [KGS15; Arn+15] and a tool such as [Arn+13] helps with the analysis. Probabilities of occurrence for events is heavily used in the literature, for example with stochastic games [ANP16] and attack-defense diagrams [Her+16] and stochastic Petri nets [Dal+06]. Timed automata extended with costs have been used to express attack trees [KRS15] and attack-fault trees [KS17], while [Kum+18] offers a methodology analysis using Priced Timed Automata. More details are available in the recent survey [Wid+19] on formal methods for attack tree analysis.

Regarding tool development, Uppaal has been used for automatic model transformations in [Sch+17] and in [HV06] UML sequence diagrams are manually transformed into timed automata models.

Tools such as [Gad+16b; Kum+18] offer the possibility to use combinations of known time durations, probabilities and costs. While [Boz+19] is a quite complete tool, uncertainty is not tackled the way we do with unknown parameters. [KS17] especially tackles the problem of multiple complex risk metrics and attacker profiles, in a probabilistic and timed formalism that can be computed and analyzed using stochastic model-checking [RS14] and Uppaal SMC [Dav+15]. AFTs are modeled in the Galileo format and translated with the tool ATTop [Kum+18] into stochastic timed automata [Dav+11].

However, synthesis of multidimensional parameters (time, cost for the attacker, damages for the organization...) at once for fully timed systems is not treated in the previously cited works, and these works require testing one by one a set of possible attribute values for an AFT.

In a completely different area, asynchronous hardware circuits' gates were translated into (parametric) timed automata in [Che+09]; our translation of AFTs gates into PWTAs synchronized using parallel composition shares some similarities with that approach.

Another famous extension of attack trees is attack-defense trees in order to model and figure out the best fitted countermeasures and whether it is worth to develop against an attack. Attack-defense trees are well-studied and new analysis methods are still developed [Kor+14; Gad+16a; KW18; Pet+19; Val+20], even with parameters [Ari+20]. In our approach, we propose to use parameters to help figuring out the best countermeasures.

Outline We recall attack-fault trees in Section 2. We then introduce the formalism of *parametric weighted timed automata* in Section 3. Our translation from AFTs to PWTAs is given in Section 4. Then, we describe our implementation in Section 5 and report on experiments in Section 6. Section 7 extends our work by allowing countermeasures. We conclude by discussing future works.

2 Attack-fault Trees

Attack-fault trees (AFTs) model how a safety or security goal can be refined into smaller sub-goals, represented as *gates*, until no further refinement is possible, represented as *leaves*. The leaves of the tree model are either basic component failures

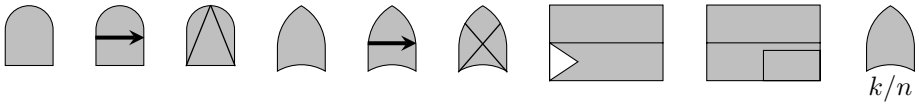


Figure 1: From left to right: AND, SAND, PAND, OR, SOR, XOR, FDEP, SPARE, VOT(k/n) gates

(BCF) or basic attack steps (BAS). Since subtrees can be shared in the literature (see e. g., [KS17]), AFTs are actually directed acyclic graphs, rather than trees. In this paper, we consider only trees without shared gates or leaves. Safety is compromised with the failure of a BCF, i. e., without any outside spark action. Security is compromised when an outside attacker causes the activation of a BAS. Following the terminology of [KS17], in this paper write that a gate or a leaf is *disrupted* if the output is true i. e., it succeeds, and *fails* otherwise. A success event (disruption) models the fact that a component (gate or leaf) is compromised i. e., the attack is successful or the component fails. In contrast, a fail event models the robustness of the component against an attacker through a BAS, or a BCF.

2.1 AFT leaves

AFT leaves are equipped with an execution time and a rich cost structure that includes the cost incurred by an attacker and damage inflicted on the organization. In contrast to [RS15; Her+16; KS17] where BCF and BAS are equipped with probability distributions, we consider both BCF and BAS as parametric time-dependent events. This allows us to compute a range of cost values, damages values and time intervals at once in order to perform operations such as optimum time values for a counter-measure while keeping damage to the organization low, and cost for the attacker high.

2.2 AFT gates

In order to model complex scenarios with multiple leaves, BCF and BAS have to be composed. For this purpose, logical gates are used that output either the propagation of a disruption, or not. Gates take as an input either leaves or outputs from gates in their subtrees. Logical gates used in AFTs are taken from both dynamic fault trees and attack trees: AND, PAND, SAND, OR, SOR, FDEP, SPARE, VOT(k/n), depicted in Fig. 1. These gates are the translatable ones in ATTop [Kum+18] from the Galileo format. We also added the XOR gate to improve our modeling capabilities.

An AND gate propagates a disruption (i. e.,, it synchronizes a success event [KS17]) if all of its children are disrupted, regardless of the order of disruption. Children are activated initially by the AND. Children of a SAND gate are activated sequentially from left to right. After the success (disruption) of the leftmost child, the second left most child is activated, and so on until the disruption of rightmost child. If *all* children are disrupted, the SAND gate is disrupted. However, if any child fails (to be disrupted), the SAND gate directly fails. SAND gate is a specific gate of attack trees. Compared with a SAND gate, all children of a PAND gate are activated initially when the PAND gate is activated. The rest of the execution is similar to a SAND gate, and propagates a disruption if all children are disrupted from left to right (which in contrast is not mandatory for an AND gate), otherwise the PAND gate fails.

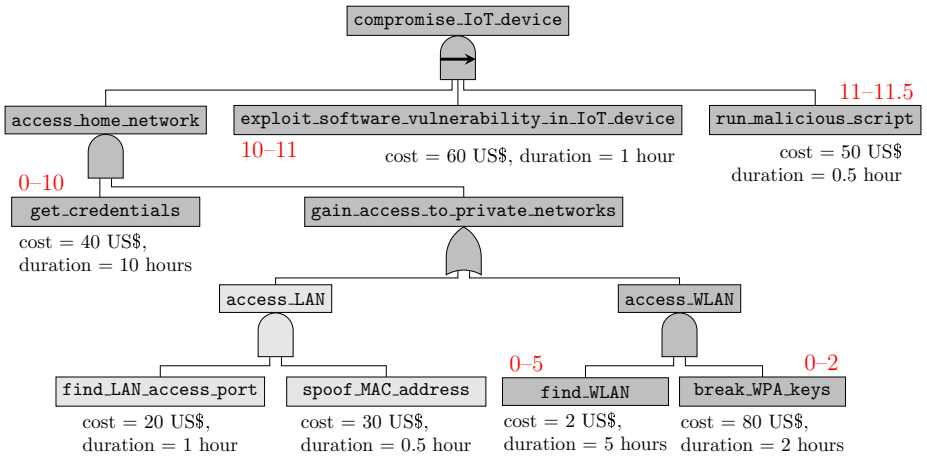


Figure 2: Attack Tree modeling the compromise of an IoT device from [Sch+17]. Leaves are equipped with the cost and time required to execute the corresponding step. The parts of the tree attacked in a successful attack are indicated by a darker color, with start and end times for the steps in this attack denoted in red.

An OR gate propagates a disruption if at least one of its children is disrupted. Children are activated initially by the OR gate. Similarly to a SAND gate, children of a SOR gate are activated sequentially after the termination of the previous one and from left to right. It propagates a disruption *when* one of its children is disrupted, otherwise if all children fail the SOR gate fails. A XOR gate propagates a disruption if one of its children is disrupted and the other one fails.

A FDEP (functional dependency) gate consists of a trigger event and several dependent events, and is a specific gate of fault trees (named TRIGGER gate in [Her+16]). When the trigger event occurs, all its dependent BCF events are disrupted (i. e., the failure of the power supply automatically deactivate the alarm and security cameras, therefore the BCFs are successful).

A SPARE gate is similar to the SAND, but is a specific gate for fault events while the SAND gate is used for attack events. A SPARE gate consists of one primary BCF and several secondary BCF which are activated sequentially. If the primary BCF is disrupted (i. e., the component fails), a secondary becomes primary. If no BCFs are left (they all are disrupted), the SPARE gate propagates a disruption.

A VOT(k/n) gate is similar to an OR gate and consists of $n \in \mathbb{N}$ children initially activated. A VOT(k/n) gate is disrupted when k of its n children are disrupted.

3 Parametric weighted timed automata

Let $\mathbb{N}$, $\mathbb{Q}$, $\mathbb{Q}_+$, and $\mathbb{R}_+$ denote the set of non-negative integers, rationals, non-negative rationals and non-negative reals, respectively.

We assume a set $\mathbb{C} = \{x_1, \ldots, x_H\}$ of *clocks*, i. e., real-valued variables that evolve at the same rate. A *clock valuation* is a function $\nu : \mathbb{C} \to \mathbb{R}_+$. We write $\vec{0}_{\mathbb{C}}$ for the

clock valuation assigning 0 to all clocks. Given $d \in \mathbb{R}_+$, $\nu + d$ denotes the valuation s.t. $(\nu + d)(x) = \nu(x) + d$, for all $x \in \mathbb{C}$. Given $R \subseteq \mathbb{C}$, we define the *reset* of a valuation ν, denoted by $[\nu]_R$, as follows: $[\nu]_R(x) = 0$ if $x \in R$, and $[\nu]_R(x) = \nu(x)$ otherwise.

We assume a set $\mathbb{TP} = \{p_1, \ldots, p_J\}$ of *timing parameters*, i. e., unknown timing constants. A *timing parameter valuation* tv is a function $tv : \mathbb{TP} \to \mathbb{Q}_+$. We assume $\bowtie \in \{<, \leq, =, \geq, >\}$. A *guard* g is a constraint over $\mathbb{C} \cup \mathbb{TP}$ defined by a conjunction of inequalities of the form $x \bowtie d$, or $x \bowtie p$ with $x \in \mathbb{C}$, $d \in \mathbb{N}$ and $p \in \mathbb{TP}$. Given g, we write $\nu \models tv(g)$ if the expression obtained by replacing each x with $\nu(x)$ and each p with $tv(p)$ in g evaluates to true.

We assume a set $\mathbb{W} = \{w_1, \ldots, w_M\}$ of *weights*. A *weight valuation* μ is a function $\mu : \mathbb{W} \to \mathbb{Q}$. We write $\vec{0}_{\mathbb{W}}$ for the weight valuation assigning 0 to all weights. We assume a set $\mathbb{WP} = \{q_1, \ldots, q_N\}$ of *weight parameters*, i. e., unknown weight constants. A *weight parameter valuation* wv is a function $wv : \mathbb{WP} \to \mathbb{Q}$.[1] A linear arithmetic expression over $\mathbb{W} \cup \mathbb{WP}$ is $\sum_i a_i w_i + \sum_j b_j q_j + c$, where $w_i \in \mathbb{W}$, $q_j \in \mathbb{WP}$ and $a_i, b_j, c \in \mathbb{Q}$. Let $\mathcal{LA}(\mathbb{W} \cup \mathbb{WP})$ denote the set of arithmetic expressions over $\mathbb{W}$ and $\mathbb{WP}$. A parametric weight update is a partial function $\alpha : \mathbb{W} \nrightarrow \mathcal{LA}(\mathbb{W} \cup \mathbb{WP})$. That is, we can assign a weight to an arithmetic expression of parametric weights and other weight values, and rational constants. Given a weight valuation μ, a parametric weight update α and a weight parameter valuation wv, we need an evaluation function $eval_{wv}(\alpha, \mu)$ returning a weight valuation, and defined as follows:

$$eval_{wv}(\alpha, \mu)(w) = \begin{cases} \mu(w) & \text{if } \alpha(w) \text{ is undefined} \\ \mu(wv(\alpha(w))) & \text{otherwise} \end{cases}$$

where $\mu(wv(\alpha(w)))$ denotes the replacement within the linear arithmetic expression $\alpha(w)$ of all occurrences of a weight parameter q_i by $wv(q_i)$, and of a weight variable w_j with its current value $\mu(w_j)$. Observe that this replacement gives a rational constant, therefore $eval_{wv}(\alpha, \mu)$ is indeed a weight valuation $\mathbb{W} \to \mathbb{Q}$. That is, $eval_{wv}(\alpha, \mu)$ computes the new (non-parametric) weight valuation obtained after applying to μ the partial function α valuated with wv.

Parametric timed automata (PTA) extend timed automata [AD94] with timing parameters within guards and invariants in place of integer constants [AHV93]. We extend further PTA with (discrete) rational-valued *weight parameters*, giving birth to parametric weighted timed automata (PWTA).

Definition 1. A parametric weighted timed automaton (PWTA) $\mathcal{A}$ is a tuple $\mathcal{A} = (\Sigma, L, l_0, F, \mathbb{C}, \mathbb{TP}, \mathbb{W}, \mathbb{WP}, I, E)$, where:

1. Σ is a finite set of synchronization actions,
2. L is a finite set of locations,
3. $l_0 \in L$ is the initial location,
4. $F \subseteq L$ is the set of accepting locations,
5. $\mathbb{C}$ is a finite set of clocks,
6. $\mathbb{TP}$ is a finite set of timing parameters,
7. $\mathbb{W}$ is a finite set of weights,
8. $\mathbb{WP}$ is a finite set of weight parameters,

[1] Observe that, in contrast to timing parameters that should be non-negative (which is usual for *parametric timed automata*), our weight parameters may be negative.

9. I is the invariant, assigning to every $l \in L$ a guard $I(l)$,
10. E is a finite set of edges $e = (l, g, a, R, \alpha, l')$ where $l, l' \in L$ are the source and target locations, g is a guard, $a \in \Sigma$, $R \subseteq \mathbb{C}$ is a set of clocks to be reset, and $\alpha : \mathbb{W} \nrightarrow \mathcal{LA}(\mathbb{W} \cup \mathbb{WP})$ is a parametric weight update.

Given a timing parameter valuation tv and a weight parameter valuation wv, we denote by $tv|wv(\mathcal{A})$ the non-parametric structure where all occurrences of a timing parameter p_i have been replaced by $tv(p_i)$, and all occurrences of a weight parameter q_j have been replaced by $wv(q_j)$. The resulting structure can be seen as an extension of a parametric weighted/priced timed automaton [Beh+01; ALP04] with only rational weights on edges.[2] However, our structure goes beyond a simple parametric extensions of weighted/priced timed automata, for two reasons:

1. we allow multiple weights;
2. we allow to not only *increment* weight values over a path, but also perform more complex operations on that weight, notably incrementing it *with another weight value*, which is clearly not possible in [Beh+01; ALP04].

Note that, if we restrict our parametric weight update function to expressions of the form $\alpha(w_i) = w_i + z$, where z is either a weight parameter or a rational constant, then our formalism is exactly the parametric extension of (the discrete "switch" weight part of) [Beh+01; ALP04].[3]

In addition, our formalism shares some similarities with the statically parametric timed automata of [Wan00], where timed automata are extended with parameters that can only be used in guards, but not compared to clocks. In contrast, our weight parameters can only be used in updates, and not in guards; in addition, we also feature the timing parameters of [AHV93] that can be compared to clocks.

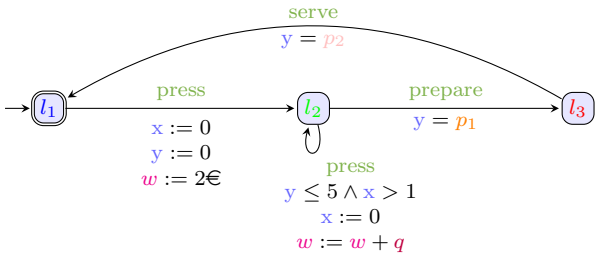


Figure 3: A PWTA modeling a coffee machine

Example 1. In the PWTA in Fig. 3, we have the following elements: $L = \{l_1, l_2, l_3\}$, $l_0 = l_1$ (also the unique element of F), $\mathbb{C} = \{x, y\}$, and $\Sigma = \{\text{press}, \text{prepare}, \text{serve}\}$, with the set $\mathbb{TP} = \{p_1, p_2\}$ and weights $\mathbb{W} = \{w\}$, $\mathbb{WP} = \{q\}$. There are four edges:

[2] In [ALP04] cost is defined as the sum of each discrete cost on transitions (*switch* cost) plus the time spent in a location multiplied by an integer rate (*duration* cost), resulting in a rational value. Here, we omit the duration costs.

[3] Technically, as weighted/priced timed automata use integer constants, a rescaling of the constants is necessary: by multiplying all constants in $tv|wv(\mathcal{A})$ by the least common multiple of their denominators, we obtain an equivalent (integer-valued) weighted/priced timed automata.

- $e_1 = \langle l_1, g, a, R, l_2 \rangle$ where R sets both x, y to 0, α is $w = 2€$,
- $e_2 = \langle l_2, g, a, R, l_2 \rangle$ where g is $\mathrm{y} \leq 5 \wedge \mathrm{x} > 1$ and R sets x to 0, α is $w := w + q$,
- $e_3 = \langle l_2, g, a, R, l_3 \rangle$ where g is $\mathrm{y} = p_1$ and
- $e_4 = \langle l_3, g, a, R, l_1 \rangle$ where g is $\mathrm{y} = p_2$.

Let us now define the concrete semantics of PWTA as the union over all timing parameter and weight parameter valuations.

Definition 2 (Semantics of a valuated PWTA)**.** Given a PWTA $\mathcal{A} = (\Sigma, L, l_0, F, \mathbb{C}, \mathbb{TP}, \mathbb{W}, \mathbb{WP}, I, E)$, a timing parameter valuation tv, and a weight parameter valuation wv, the semantics of $tv|wv(\mathcal{A})$ is given by the timed transition system (TTS) $(S, s_0, \to)$, with

- $S = \{(l, \nu, \mu) \in L \times \mathbb{R}_+^H \times \mathbb{Q}^M \mid \nu \models tv(I(l))\}$,
- $s_0 = (l_0, \vec{0}_{\mathbb{C}}, \vec{0}_{\mathbb{W}})$,
- $\to$ consists of the discrete and (continuous) delay transition relations:
  1. discrete transitions: $(l, \nu, \mu) \overset{e}{\mapsto} (l', \nu', \mu')$, if $(l, \nu, \mu), (l', \nu', \mu') \in S$, and there exists $e = (l, g, a, R, \alpha, l') \in E$, such that $\nu \models tv(g)$, $\nu' = [\nu]_R$, and $\mu' = eval_{wv}(\alpha, \mu)(w)$;
  2. delay transitions: $(l, \nu, \mu) \overset{d}{\mapsto} (l, \nu + d, \mu)$, with $d \in \mathbb{R}_+$, if $\forall d' \in [0, d], (l, \nu + d', \mu) \in S$.

That is, a state is a triple made of the current location, the current (non-parametric) clock valuation, and the current (non-parametric) weight valuation. The clock valuations evolve naturally as in timed automata, while the current weight evolves according to the weight update function.

Moreover we write $(l, \nu, \mu) \overset{(e,d)}{\longrightarrow} (l', \nu', \mu')$ for a combination of a delay and discrete transition if $\exists \nu'' : (l, \nu, \mu) \overset{d}{\mapsto} (l, \nu'', \mu) \overset{e}{\mapsto} (l', \nu', \mu')$. Given $tv|wv(\mathcal{A})$ with concrete semantics $(S, s_0, \to)$, we refer to the states of S as the *concrete states* of $tv|wv(\mathcal{A})$. A *run* of $tv|wv(\mathcal{A})$ is an alternating sequence of concrete states of $tv|wv(\mathcal{A})$ and pairs of edges and delays starting from the initial state s_0 of the form $s_0, (e_0, d_0), s_1, \cdots$ with $i = 0, 1, \ldots$, $e_i \in E$, $d_i \in \mathbb{R}_+$ and $(s_i, e_i, s_{i+1}) \in \to$.

Example 2. A concrete execution of the PWTA $tv|wv(\mathcal{A})$ of Example 1 with $w = 2€$, $wv(q) = 0.5€$, $tv(p_1) = 5$ and $tv(p_2) = 8$ is

$(l_1, (0,0), (0)) \overset{(\text{press},2)}{\longrightarrow} (l_2, (0,0), (2)) \overset{(\text{press},1.5)}{\longrightarrow} (l_2, (0,1.5), (2.5)) \overset{(\text{press},1)}{\longrightarrow}$
$(l_2, (0,2.5), (3)) \overset{(\text{prepare},2.5)}{\longrightarrow} (l_3, (2.5,5), (3)) \overset{(\text{serve},3)}{\longrightarrow} (l_1, (5.5,8), (2.5))$.

Note that no coffee can be served if $tv(p_1) = 8$ and $tv(p_2) = 5$.

Remark 1. Despite the name of *weights* (justified by our context of measuring costs and damages), our parametric weights are in fact sufficiently expressive to encode parametric (rational-valued) data.

4 Translation of AFTs to PTAs

4.1 Overview of the translation

We will model an attack-fault tree using a network of PWTAs that will synchronize along actions (using the usual composition semantics). Each gate and each leaf (i. e.,

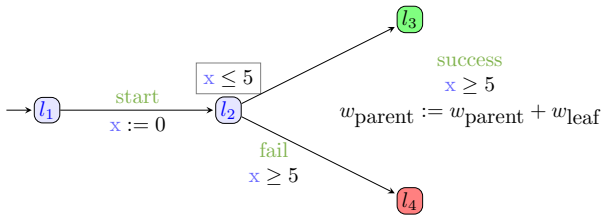


Figure 4: PWTA translation of leaf that can reach the success location in exactly 5 units of time, and of weight w_{leaf}

BAS or BCF) will be modeled as a PWTA. Leaves PWTA have a duration and a weight, while gates PWTA store the weight value of their children to forward it to their parents. Therefore, each gate PWTA maintains its own weight, and its value will be added to that of their parents in case of success (thanks to the parametric weight update).

All gates and leaves PWTAs initially synchronize their start action—referred as *activation* in this paper—, and end with either a success or fail synchronization action. After gates synchronize their start action, they synchronize the start action of their children.

Intuitively, the process is top-bottom-top: the top level gate PWTA activates its children, which themselves activate their children (if any), and so on until the leaves PWTAs at the bottom of the attack-fault tree. Once a leaf PWTA terminates, it synchronizes either its success or fail action. In case of success, the leaf PWTA forwards its weight value to its parent, where this value is stored. When its parent gate PWTA terminates, the gate PWTA synchronizes either a success or a fail action. In case of success, the gate PWTA forwards its weight value to its parent, and so on until the top-level gate PWTA terminates.

If the top-level PWTA terminates in its success location, the attack is successful. We apply the reachability synthesis algorithm of PTAs on the success location in the top-level PWTA, that is, we synthesize all valuations for which this location is reachable: this gives us the success conditions of an attack. The set of constraints on time and weight (such as cost for the attacker, damages for the organization) that allowed this attack to be successful are output by this analysis.

As a running example, we consider the attack tree in Fig. 2 taken from [Sch+17].

4.2 Translation of leaves

A BAS/BCF is modeled as a PWTA with clocks and weights (see Fig. 4). Note that in real life while a BAS needs to success so the attack is possibly successful, a BCF needs to fail in order to propagate a disruption (as in basic component *failure*). However we consider in our models that both BAS and BCF need to reach the success location. there are two paths in a BAS/BCF PWTA, one that reaches the success location and one that reaches the fail location. In case of success, its weight is forwarded to and stored in its parent gate.

Example 3. The translation of leaf `find_WLAN` of Fig. 2 is given in Fig. 6c. To express the leaf `find_WLAN` we use four locations, one clock $x4$. The first step is to activate the basic attack step using the synchronization action *launchFindWLAN*.

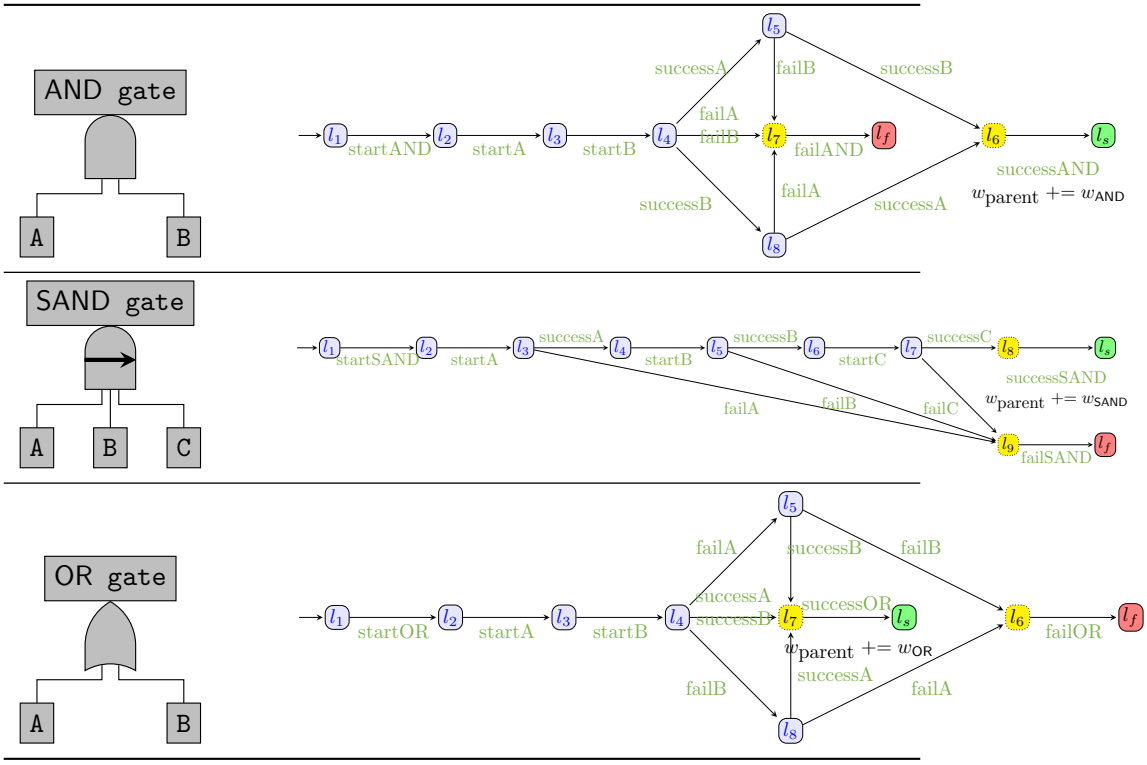


Table 1: Translation rules of AND, SAND and OR gates to PWTA

Once activated, and at most five units of time after (modeled by the invariant $5 \geq x4$ and the guard $x4 \geq 5$) it can either success with the action *successFindWLAN* or fail with the action *failFindWLAN*. If the success state is reached, the weight of its parent gate is increased by its own weight 10.

4.3 Translation of gates

Concrete translations of SAND, AND, OR gates are given in Table 1 (yellow locations denote urgency: time cannot elapse). We describe them and give examples in the following. Other gates are similar. Note that our translations are parameterized versions of the gates of [KRS15; KS17] which are convenient to capture the behavior of the gates with automata in general.

AND Recall that an AND gate is disrupted if all of its children are disrupted. It activates all of its children then waits for their disruptions regardless of the order of the successes. At any moment if one fails, the AND gate fails. If the success action is synchronized, its parent weight w_{parent} is updated: the weight w_{AND} carried by the AND gate is added to w_{parent}.

Example 4. We give in Fig. 5b the PWTA corresponding to the AND gate `access_home_network` of Fig. 2. When all children are activated in the PWTA of

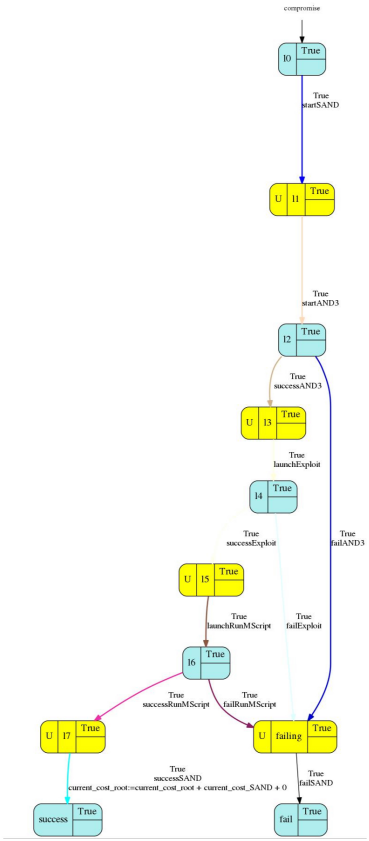
Fig. 5b, there are four paths leading to the fail state, while only two (success of the two children in any order) leading to the success state. *startAND3* launches the AND gate `access_home_network`. Both children, the BAS `get_credential` and the OR gate `gain_access_to_private_networks` are activated with the synchronization of the actions *launchGetCred* and *startOR*. Unlike the SAND gate, an AND gate waits for any of its child to synchronize a success action. If *successGetCred* is synchronized, it then will wait for *successOR* to go to the location success. If *failGetCred* is synchronized, the automaton will go to the location failing. When waiting for the action *successOR*, if *failOR* is synchronized the automaton will also go to the location failing. The other possibility (*successOR* then *successGetCred*) is similar. When in location failing it synchronizes the action *failAND3*, while if going to the location success it will synchronize the action *successAND3*. If the success state is reached, the weight of its parent gate is increased by its own weight.

SAND Recall that a SAND gate is disrupted if all its children from left to right are disrupted sequentially from left to right. It activates its leftmost child then waits for its success or failure, then activates its second leftmost child and so on. If the rightmost child succeeds, the SAND gate is disrupted. If one child fails, the SAND gate fails. For a SAND gate modeled as a PWTA with n children, there is only one path leading to the success state, while there are n paths leading to the fail state (one from each child). If the success action is synchronized, its parent weight w_{parent} is updated: the weight w_{SAND} carried by the SAND gate is added to w_{parent}.

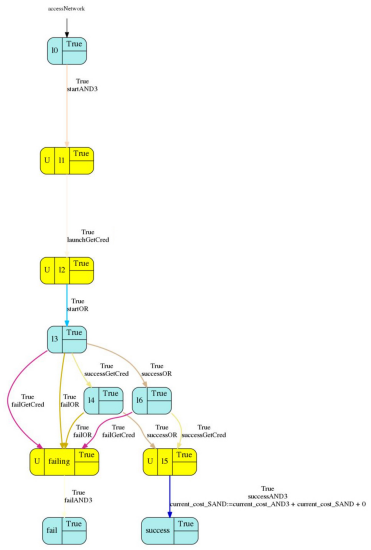
Example 5. The top event of the attack tree in Fig. 2 is a SAND gate. We give the PWTA corresponding to this SAND in Fig. 5a. It synchronizes the action *startSAND*. Then it activates its leftmost child `access_home_network` with the action *startAND3*, which is an AND gate. If the action *successAND3* is synchronized, its second leftmost child is activated with the action *launchExploit*. If the action *successExploit* is synchronized, its third and last child is activated with the action *launchRunMScript*. If the action *successRunMScript* is synchronized, the action *successSAND* is synchronized. At any moment, if one of its children fail and an action *failAND3*, *failExploit* or *failRunMscript* is synchronized the automaton goes to the location failing where the action *failSAND* is synchronized. If the success state is reached, the weight of its parent gate is increased by its own weight.

OR An OR gate initially activates all of its children. An OR gate is disrupted if at least one of its children is disrupted, and fails if all of its children fail. Therefore in the case of two children, one child can fail and the OR gate still propagates a disruption if the other one succeeds right after. However, if one child succeeds no need to wait for the second one and the success action of the OR gate is synchronized. If the success action is synchronized, its parent weight w_{parent} is updated: the weight w_{OR} carried by the OR gate is added to w_{parent}.

Example 6. The PWTA translating the only OR of Fig. 2 (given in Fig. 6a) activates all of its children, then waits for one to succeed, regardless of the order. Afterwards, whatever happens leads to the success state. If one child fails, then the other has to succeed, otherwise the OR fails. Therefore there is six possible paths to the success state, while there are two paths to the fail state (failure of both children in any order). *startOR* launches the OR gate `gain_access_to_private_networks` which activates its two children using the actions *startAND1* and *startAND2* which activates the AND



(a) Translation of the top-level SAND gate



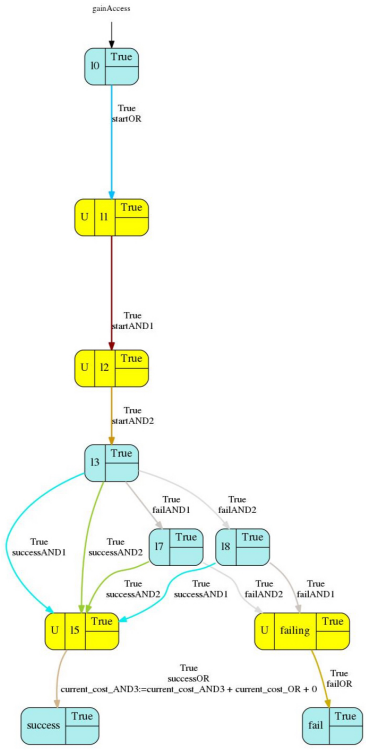
(b) Translation of the AND gate `access_home_network`

Figure 5: SAND and AND gate

`access_LAN` and AND `access_WLAN`. Only one action *successAND1* or *successAN2* is needed to be synchronized so the automaton goes to the location success regardless of which action is synchronized afterwards. Then it synchronizes the action *successOR*. If at first the action *failAND1* (resp. *failAND2*) is synchronized, then *successAND2* (resp. *successAND1*) has to be synchronized afterwards in order to reach the location success. Otherwise, if *failAND2* (resp. *failAND1*) is synchronized, the automaton will go to the location failing and then synchronize the action *failOR*. If the success state is reached, the weight of its parent gate is increased by its own weight.

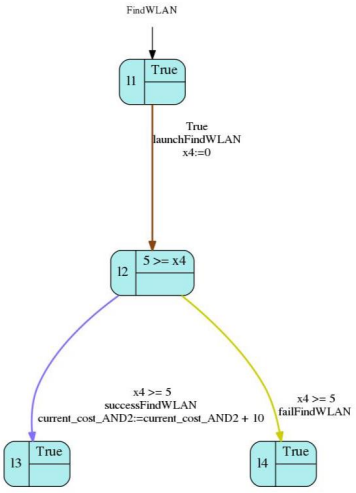
4.4 Top-level automaton

Finally, we need to create an automaton that will activate the first top-event gate of the AFT. We call it rootTA. This PWTA is the one that starts the chain reaction by activating the top-event PWTA gate, which at its turn will activate its own children and so on. It waits for the success or fail action of this PWTA gate. In case of



(a) Translation of the OR gate `gain_access_to_private_networks`

(b) OR



(c) BAS translation of `find_WLAN`

Figure 6: OR gate and BAS

success, its weight has been updated with the total weight value of the execution forwarded by the top-event gate PWTA. This bottom-to-top addition stores in the weight *current_cost_root* the total weight of the attack. The rootTA also stores the total time spent since the first activation of the top-event PWTA (using an extra clock and parameter).

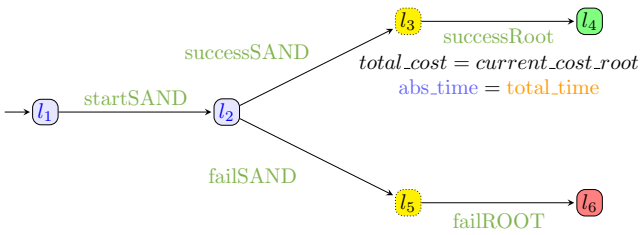


Figure 7: The rootTA

Example 7. We give in Fig. 7 the top-level PWTA for the AFT in Fig. 2. It is very similar to a leaf PWTA. It activates the top-level gate PWTA, then waits for its success or fail action. If the success action is synchronized, its weight has been updated to the total weight value of the execution and is checked against an additional parameter *total_cost* so IMITATOR outputs this *current_cost_root* value. Likewise, the clock abs_time which is never reset since the activation of rootTA is checked against a timing parameter total_time. Therefore IMITATOR outputs the total time of execution.

5 Implementation of the translation

5.1 IMITATOR

IMITATOR [And21] is a parametric model checker taking as input networks of parametric timed automata extended with synchronization, stopwatches and discrete variables. IMITATOR supports global (shared) discrete rational-valued variables, that can be either *concrete* (in which case they are syntactic sugar for an unbounded number of locations), or *symbolic*, in which case they can be updated to or compared with parameters. While IMITATOR technically considers a single type of parameters (where symbolic variables can be compared or even updated to *timing* parameters), our weight parameters are never compared to timing parameters, and this setting can be considered as a subclass of the IMITATOR expressiveness.

IMITATOR implements several synthesis algorithms, notably reachability synthesis (EFsynth), that attempts to synthesize all parameter valuations for which a given location is reachable—which is the algorithm we use here. IMITATOR relies on the symbolic semantics of parametric timed automata (see e. g., [And+09; JLR15]), where symbolic states are made of a discrete location, and a constraint over the clocks and parameters. The weight parameters are added to this symbolic semantics in a straightforward manner, with symbolic states enriched with linear constraints over weight parameters.

Note that, while parametric timed automata are highly undecidable (see [And19] for a survey), and while our parametric extension adds a new layer of complexity, all analyses terminate with an exact result (sound and complete) because our models are *acyclic*: our AFTs are trees, and their translation yields structurally acyclic PWTAs. As a consequence, the symbolic semantics of these PWTAs can be represented as a finite structure, and the analysis is guaranteed to terminate.

5.2 Translation from AFTs to PWTAs

The translation from AFTs to PWTAs was implemented within the framework of ATTop [Sch+17]. The existing software ATTop can take as input a Galileo formatted file. This format is easy to use and to understand. The code in Fig. 8 expresses an attack-fault tree of one OR gate named `A`, with two children `B` and `C`. The BAS `B` takes between 50 and 100 units of time to terminate, and costs \$50 to the attacker. The BAS `C` takes between 30 and 70 units of time to terminate, and costs \$30 to the attacker.

ATTop takes as an input a Galileo file and parses it to represent it as an attack-fault tree meta-model (ATMM) (see [Sch+17, Section 3], and Fig. 9 for a screenshot of the tool).

Then, different translations are available: one quite interesting is the translation into an Uppaal file, for instance a network of stochastic timed automata [KS17]. ATTop takes the ATMM and translates it in its Uppaal meta-model, then serializes it into an

```
toplevel "A";
"A" or "B" "C";
"B" mintime=50 maxtime=100 cost=50;
"C" mintime=30 maxtime=70 cost=30;
```

Figure 8: Example of Galileo attack tree

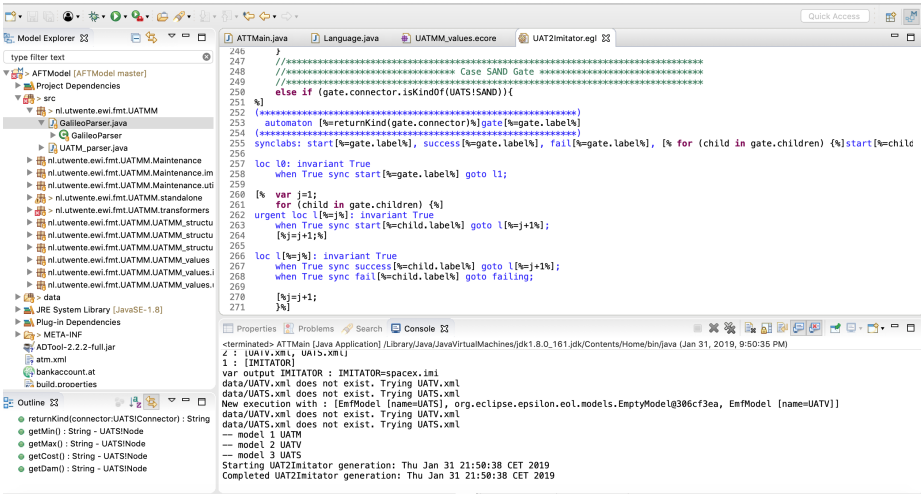


Figure 9: Screenshot of the tool ATTop after the translation of the SpaceX AFT

Uppaal formatted file. In our approach we directly translate the representation of the ATMM into an IMITATOR formatted file, using the Epsilon Generation Language (EGL) [Ros+08]. This translation is a very efficient way to obtain AFTs modeled using PWTAs: designing manually a PWTA model from an AFT is very tedious to achieve, while defining an AFT within the Galileo syntax is simple.

Once the PWTA obtained, we synthesize using IMITATOR all parameter valuations for which the success location of the rootTA can be reached (using EFsynth). These sets of parameter values will help us to determine attack and fault scenarios in the following section.

6 Case studies

As a proof of concept, we apply our approach to an attack tree from the literature and an original attack-fault tree. Experiments were conducted with IMITATOR 2.10.4 "Butter Jellyfish",[4] on a 2.4 GHz Intel Core i5 processor with 2 GiB of RAM in a VirtualBox environment. Computation times of parameter values range from 1 to 9 seconds with four parameters.

[4] Sources, binaries, models and results are available at imitator.fr/static/ACSD19PAT/ and 10.5281/zenodo.5062314

6.1 Compromising an IoT device

We apply our approach to the AFT depicted in Fig. 2 taken from [Sch+17]. We choose to parametrize the cost of finding a LAN access point (CostFindLAN_AP) and the maximum amount of time to break WPA keys (tMax_Break) of the AFT. This configuration will describe which attack (WLAN or LAN) is smarter for the attacker, depending on their resources: finding a LAN access point can be difficult depending on the infrastructure security and perhaps social engineering is needed. However, if the attacker does not have enough resources but a large amount of time (s)he can spend time trying to break WPA keys. IMITATOR computes several constraints on these parameters such that the attack is successful.

Different constraints are possible representing possible time and weight values s.t. an attack is possible. This is represented as a disjunction of conjunctions of constraints on parameters. For instance it can be a quick but very costly attack, or a long but cheap one; therefore different attack and fault scenarios appear. The conjunction of constraints

$$2 * \text{tMax\_Break} \geq 23 \wedge \text{CostFindLAN\_AP} \geq 0$$
$$\wedge \text{CostFindLAN\_AP} + 180 = \text{total\_cost} \wedge 2 * \text{total\_time} = 23$$

represents an attack that is very expensive for the attacker: indeed, the total cost of the attack is at least $180 and fully depends on the cost of finding a LAN access point. However, the time spent on the attack is negligible and fixed (11.5h).

In opposition, the constraint

$$2 * \text{tMax\_Break} + 3 \geq 2 * \text{total\_time}$$
$$\wedge \text{CostFindLAN\_AP} \geq 0$$
$$\wedge 2 * \text{total\_time} \geq 23 \wedge \text{total\_cost} = 232$$

shows a an attack that will last at least 11.5h—that is, the attacker does not exactly know when (s)he will break the WPA keys depending for instance of her/his computation power—but with a fixed cost of $232.

Contrarily to our initial intuition, the cost of this second attack can be high above the first one, as breaking the WPA keys is quite costly ($80) in opposition with finding a LAN access point. A smart attacker could choose, regardless of their time and resources the first attack through LAN access point.

6.2 SpaceX rocket Falcon 9 explosion

Our second case study is an adaptation of the anomaly investigation that followed the explosion of SpaceX rocket Falcon 9 in september 2016[5]. The AFT in Fig. 10 depicts the different configurations that can eventually end up with the explosion. The objective of this case-study is to show that the explosion is more likely to be accidental, due to the expensiveness of the BAS for the attacker who could attempt a sabotage.

The rocket carries a helium tank with three composite overwrapped pressure vessels (COPVs) inside. One COPV possibly had a manufacturing defect and buckles in its

[5]SpaceX anomaly update, https://www.spacex.com/news/2016/09/01/anomaly-updates

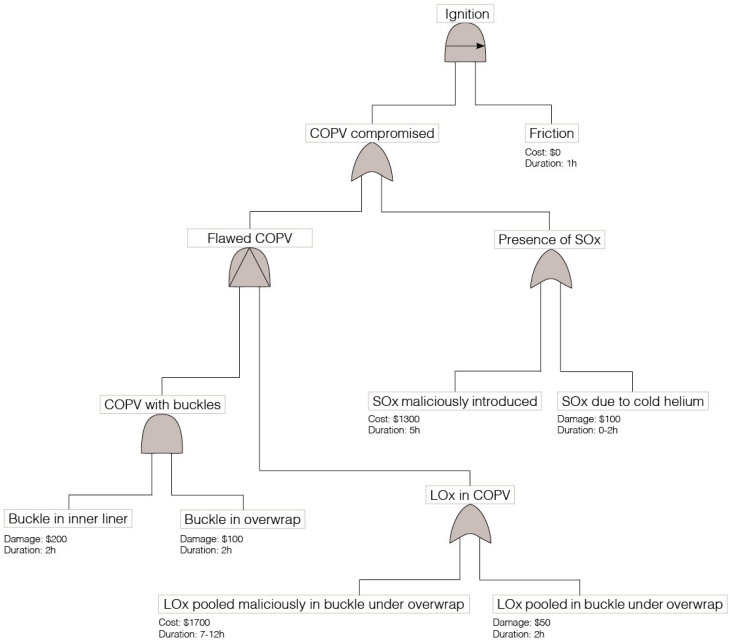


Figure 10: AFT of SpaceX rocket explosion

liner and the carbon overwrap (AND gate). Afterwards (PAND gate) liquid oxygen (LOx) can pool in these buckles and become trapped when pressurized under the carbon overwrap, resulting in a flawed COPV. An other possibility is the presence of solid oxygen (SOx) either due to the loading temperature of helium or placed here intentionally by an attacker (OR gate).

These two configurations result in a compromised COPV. When the COPV is compromised, a friction due to take-off tests can start the rocket ignition (SAND gate).

BCFs have a duration representing the time taken until the component failure. Damage is the cost for the organization for having built a defective component, or the cost induced when the component has failed. BASs have a cost for the attacker to perform the attack, and a duration for the attack to be successful.

We choose to parametrize damages induced to the manufacturing facility by damage_BuckleInInnerLiner, and the cost of pooling solid oxygen near the COPV, cost_SOXmaliciouslyIntroduced.

The constraint

$$\begin{aligned} &13 \geq \text{total\_time} \geq 8 \\ \wedge &\text{cost\_SOXmaliciouslyIntroduced} \geq 0 \\ \wedge &\text{total\_damages} \geq 100 \\ \wedge &\text{damage\_BuckleInInnerLiner} + 100 = \text{total\_damages} \\ \wedge &\text{total\_cost} = 1700 \end{aligned}$$

represents the attack using the malicious introduction of LOx between the inner liner and the carbon overwrap of the COPV. Clearly this attack is very costly ($1700) and assumes the presence of these buckles. It is highly prejudicial to SpaceX as the company may want to investigate the manufacturing facility that produces COPV components.

The other attack, represented by the constraint

$$\begin{gathered}\text{total\_time} = 6\\ \wedge \text{cost\_SOXmaliciouslyIntroduced} \geq 0\\ \wedge \text{total\_damages} = 0\\ \wedge \text{damage\_BuckleInInnerLiner} \geq 0\\ \wedge \text{total\_cost} = \text{cost\_SOXmaliciouslyIntroduced}\end{gathered}$$

shows that the cost of the attack is equal to the cost of introducing SOx near the COPV. The higher is the parameter cost_SOXmaliciouslyIntroduced, the higher is the cost of the attack. We may assume this cost is high enough as SpaceX surely secured its launch complex. Otherwise, an efficient counter-measure would be to find means to increase this cost for the attacker.

The constraint

$$\begin{gathered}\text{cost\_SOXmaliciouslyIntroduced} \geq 0\\ \wedge \text{total\_damage} \geq 150\\ \wedge \text{damage\_BuckleInInnerLiner} + 150 \geq \text{total\_damage}\\ \wedge \text{total\_time} = 3 \wedge \text{total\_cost} = 0\end{gathered}$$

represents the fact that buckles in the inner liner and in the carbon overwrap of the COPV, and then LOx pooled under the overwrap, lead to a complete failure of the system, i. e., the rocket explodes. In this scenario, there is in all likelihood no attacker. However, damages for the manufacturing facility can be huge if it is flawed: SpaceX should probably investigate in their manufacturing facilities in order to prevent the production of other flawed components.

Finally, the constraint

$$\begin{gathered}\text{cost\_SOXmaliciouslyIntroduced} \geq 0\\ \wedge \text{total\_damage} = 100\\ \wedge \text{damage\_BuckleInInnerLiner} \geq 0\\ \wedge 3 \geq \text{total\_time} \geq 1 \wedge \text{total\_cost} = 0\end{gathered}$$

shows that the explosion can be provoked by the presence of SOx due to cold helium. This case is possible without any attacker or component failure and is therefore fully accidental. No damages are caused to SpaceX (excepted the cost of the unusable rocket) or its suppliers.

These scenarios indicate that the rocket explosion is more likely to be accidental, as the cost in both scenarios where there is an attacker is very high. However, the worst case indicates that SpaceX should investigate their production lines to prevent other flawed components, as well as the presence of an attacker.

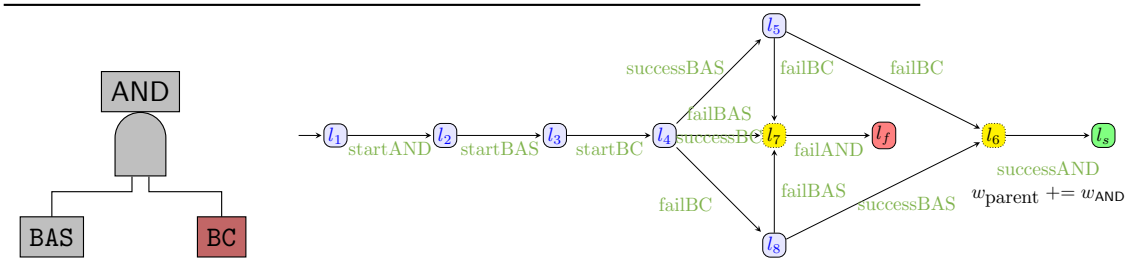


Table 2: Translation rules of the defense AND gate to PWTA

7 Counter-measures

Basic attack steps threaten the security of cyber-physical systems and their impact can be expressed and analyzed using the techniques presented in the previous sections. However, in security, it is of crucial interest to be able to find and especially define counter-measures to these attacks.

7.1 Automatically add counter-measures in AFTs

One possibility is to add a *defender* whose purpose is to counteract the attacker. This has been the subject of several papers related to *attack-defense trees* [Kor+14; Gad+16a]. In such scenarios, the defender is opposed to the attacker similarly to a parent node. Once an attack is realized, i. e., a basic attack step is disrupted, the disruption may be blocked by the defender: the success of the attack is propagated to the defender node at first, before being propagated to a logical gate. The defender acts similarly to a firewall, trying to block the attacks instead of propagating to the rest of the system. We propose a slightly different approach that emphasizes the concurrency between an attacker and a defender, similarly to [Her+16], where there is no parameter, and to [Ari+20] for a parametric version. Compared to the aforementioned works, we add parameters that we use to directly synthesize the different costs of a defense, as well as the possibility to start events and counter-measures simultaneously that evolve at possibly different time rates. In the meantime, we do not provide a new semantics definition here. This also exposes the limits of our model, where new semantics—notably for the gates—must be formally defined based for example on the use of the model, i. e., the goal of the tree, the damages we would like to compute and optimize. Concurrency is often met in real-life situations: it is possible that a defender tries to counteract an attacker in real time, during the attack. We choose here to consider an attacker and a defender as two tasks running concurrently. Both the attacker and the defender can be parameterized. Each basic attack step will be split into a parent AND gate with two children. Table 2 depicts the translation of these new elements that model the attack. One of the two is the basic attack step and remains as is. The success location in this basic attack step still models the success of the attack. The second event will be called *basic countermeasure* (BC). It is the same leaf as the basic attack step and the basic component failure, with an initial location, a success and a failure location. However, in the case of the basic countermeasure, the difference is that the success location if reached will not be propagated to its parent AND gate. In fact, the basic countermeasure will be activated if it fails.

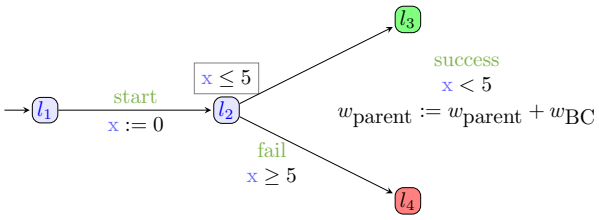


Figure 11: PWTA translation of a basic countermeasure against the leaf of Fig. 4 has to succeed within 5 units of time and of weight w_{BC}

Both basic countermeasure and basic attack step will be started at the same time. They are designed similarly and can reach their respective success or failure location according to dependent timing constraints: the key point is that the countermeasure has to succeed before the basic attack step succeeds. That is, if an attack has to succeed in exactly 5 units of time, the countermeasure has to succeed in strictly less than 5 units of time. Otherwise, the countermeasure fails, regardless of whether the attack is successful or not. Moreover, if the CM is successful, then it inhibits the concurrent BAS. The main difference with [Ari+20] is that we allow the possibility of time elapsing at a different rate for the defender than for the attacker. For example, a weak defender may see its time clock being divided by 2, i. e., the clock is evolving at 0.5 times the rate of the regular attacker clock. Conversely, a highly skilled defender may see its time clock evolving at twice the rate of the regular attacker clock.

Fig. 11 presents the PWTA working as a countermeasure against the event modeled in Fig. 4. Recall that this event can succeed in 5 units of time exactly. According to this statement, the countermeasure of Fig. 11 has to succeed in *strictly less* than 5 units of time, otherwise the only remaining path leads to a failure of the countermeasure. Similarly to basic attack steps and basic component failures, the basic countermeasure has a *cost* for the defender. It is modeled as a weight in the PWTA and written w_{BC} in Fig. 11. In scenario analyses such as the ones presented in Section 6, this *cost* can be considered as a *damage* for the organization. Even if a BC fails, it causes damages to the organization. It represents the cost of preparing and executing a counter-measure. Consider Fig. 2: adding a countermeasure to the basic attack step break_WPA_keys has potentially a financial cost for the defender who owns the IoT device. Therefore in case of success one can consider this additional cost as damage to the organization.

Both events are combined in an AND gate as the one presented in Table 2. Similarly to our previous definition of the AND gate, children's weights are forwarded to the parent gate, and the parent AND gate is disrupted if the basic attack step succeeds and the basic countermeasure fails. If it is disrupted, it forwards its own weight to its parent, as described in the previous sections.

This procedure to add a basic countermeasure against a basic attack step becomes more interesting when timing deadlines of success and failures are parameterized. We explain the implementation of our modifications in the following section.

7.2 Compromising a bank account

We propose here a small case study of a parametric attack tree with a parametric counter-measure.

Implementation. The translation from AFTs to PWTAs was implemented within the framework of ATTop [Sch+17] as described in Section 5.2. ATTop can take as input a Galileo formatted file, and with our implementation it is sufficient to signal to the software, with a boolean variable, our willingness to add a counter-measure to a BAS. If so, the BAS is automatically split into a new AND gate, a new CM and the original BAS. Still as in Section 5.2 we obtained a PWTA in the IMITATOR format.

Experiment. The experiment was conducted with the same equipment as in Section 6 and computation times of parameter values range from 1 to 3 seconds with four/five parameters.

Fig. 12 depicts a minimal attack to access a bank account. Either the hacker exploits a vulnerability in the operating system (OS) of the bank terminal, or finds a remote access to the browser of the account owner. We choose to parametrize damages induced to the bank by the attack damage_access_browser.

We automatically translate the parametric attack tree in a network of PWTAs and with IMITATOR we synthesize all parameter values such that there is a successful attack. IMITATOR outputs the following constraints such that an attack is possible:

The first constraint is

$$\begin{aligned} &\text{total\_time} = 90 \\ &\text{total\_damages} = 12 \\ \wedge &\text{total\_cost} = 40 \end{aligned}$$

and represents the attack that exploits a vulnerability in the bank terminal OS (left branch in Fig. 12).

The second constraint is

$$\begin{aligned} &50 \geq \text{total\_time} \geq 30 \\ \wedge &\text{damage\_access\_browser} = \text{total\_damages} \geq 0 \\ \wedge &\text{total\_cost} = 20 \end{aligned}$$

and represents the attack that obtains a remote access to the browser (right branch in Fig. 12). Damages is an unknown parameter and potentially high for the bank.

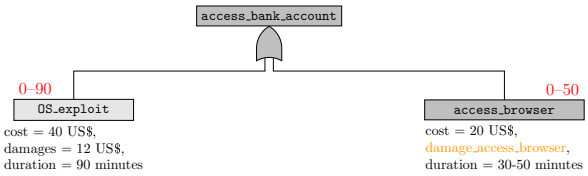


Figure 12: Parametric attack tree modeling the access of a bank account device. Leaves are equipped with the cost, damage and time required to execute the corresponding step. Start and end times for the steps in this attack denoted in red.

We now choose to define a counter-measure to the remote access to the browser and to parametrize damages induced to the bank by the counter-measure damage_CM.

We notify ATTop of the counter-measure to the remote access to the browser. The software automatically translates the parametric attack tree into a network of PWTAs with the following modifications: the initial BAS `access_browser` is transformed

into the parent AND gate `CM_access_browser` with two children, a duplicated BAS `access_browser` and a BC `counter_browser_access`. The parametric attack tree with the counter-measure we obtain is depicted in Fig. 13.

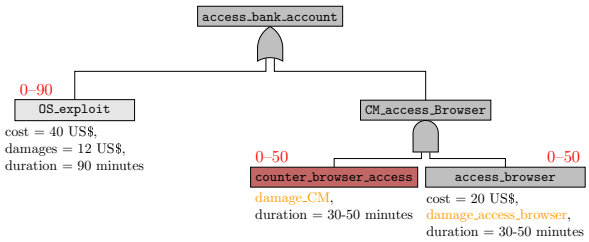


Figure 13: Transformed parametric attack tree with counter-measure modeling the access of a bank account device. Leaves are equipped with the cost (for BAS and not for BC), damage and time required to execute the corresponding step. Start and end times for the steps in this attack denoted in red.

IMITATOR outputs the same constraints for the OS exploit, and the following constraints for the browser access (right branch in Fig. 12):

$$50 \geq \text{total\_time} \geq 30$$
$$\wedge \text{total\_damages} = \text{damage\_access\_browser} + \text{damage\_CM}$$
$$\wedge \text{total\_cost} = 20$$

It is the representation of the successful remote access to the account owner's browser. In this case, a counter-measure was launched, but failed. This still has a cost for the bank, therefore is part of the total damages. Both damages are unknown parameters and therefore the total damages for the organization is possibly high.

8 Conclusion

We addressed the problem of formalizing attack-fault trees in a more abstract framework allowing to cope with parametric timings, costs and damages. We defined and implemented a translation from attack-fault trees to PWTAs (a new extension of PTAs) that can be analyzed using the IMITATOR model-checker. This translation allows us to define easily an AFT using the Galileo syntax, and obtain as an output this AFT modeled with PWTAs. Using IMITATOR, we synthesize all parameter values such that there is a successful attack and/or a system failure. Finally, obtaining a disjunction of convex sets of parameter values allows us to define different attack and fault scenarios. Therefore it help to select the most plausible scenario and the most efficient counter-measures.

We extended our basic attack step model by allowing basic parametric counter-measures to be defined. The countermeasure is run concurrently to the attack to try to block the attack in real-time.

Future works In this paper, we only considered three parameters: timing, cost and damage parameters. However, it is trivial to split these parameters into more precise

ones, such as human damages (health and insurance) and material damages caused by the attacker or the failure of the system: an attack can be cheap for the attacker but inflict many kinds of damages to the organization, as in our SpaceX case study. Thanks to the vector of weights defined in our PWTAs, this would be immediate to consider in our framework and implementation.

Moreover, adding probabilities in order to create probabilistic parametric attack-fault trees will be an interesting and challenging future work, especially to specify the probability of a countermeasure to succeed. Indeed, in our SpaceX rocket case study adding probabilities to the manufacturing defects of the COPV on top of damages inflicted to the company would strengthen considerably our formalism.

Finally, improving current modeling methodologies for risk assessment with a special focus on attack-fault trees is an ambitious future work. It would require powerful and efficient tools. Therefore we would like to perform different benchmarks and comparisons between existing tools for attack-fault tree analysis, with an emphasis on scalability for very complex models.

Acknowledgements

We are thankful to Stefano Schivo and Enno J.J. Ruijters from Twente University for their helpful advices concerning meta-models and ATTop.

References

[13] *OWASP. CISO AppSec Guide: Criteria for managing application security risks*. 2013. URL: https://owasp.org/www-pdf-archive/Owasp-ciso-guide.pdf (cit. on p. 2).

[AD94] Rajeev Alur and David L. Dill. "A theory of timed automata". In: *Theoretical Computer Science* 126.2 (Apr. 1994), pp. 183–235. ISSN: 0304-3975. DOI: 10.1016/0304-3975(94)90010-8 (cit. on p. 6).

[AHV93] Rajeev Alur, Thomas A. Henzinger, and Moshe Y. Vardi. "Parametric real-time reasoning". In: *STOC* (May 16–18, 1993). Ed. by S. Rao Kosaraju, David S. Johnson, and Alok Aggarwal. San Diego, California, United States: ACM, 1993, pp. 592–601. DOI: 10.1145/167088.167242 (cit. on pp. 2, 6, 7).

[ALP04] Rajeev Alur, Salvatore La Torre, and George J. Pappas. "Optimal paths in weighted timed automata". In: *Theoretical Computer Science* 318.3 (2004), pp. 297–322. DOI: 10.1016/j.tcs.2003.10.038 (cit. on pp. 2, 7).

[And+09] Étienne André, Thomas Chatain, Emmanuelle Encrenaz, and Laurent Fribourg. "An Inverse Method for Parametric Timed Automata". In: *International Journal of Foundations of Computer Science* 20.5 (Oct. 2009), pp. 819–836. DOI: 10.1142/S0129054109006905 (cit. on p. 14).

[And+19] Étienne André, Didier Lime, Mathias Ramparison, and Mariëlle Stoelinga. "Parametric analyses of attack-fault trees". In: *ACSD* (June 27, 2019). Ed. by Jörg Keller and Wojciech Penczek. Aachen, Germany: IEEE, 2019, pp. 33–42. DOI: 10.1109/ACSD.2019.00008 (cit. on p. 1).

[And19] Étienne André. "What's decidable about parametric timed automata?" In: *International Journal on Software Tools for Technology Transfer* 21.2 (Apr. 2019), pp. 203–219. DOI: 10.1007/s10009-017-0467-0 (cit. on p. 14).

[And21] Étienne André. "IMITATOR 3: Synthesis of timing parameters beyond decidability". In: *CAV* (July 18–23, 2021). Ed. by Rustan Leino and Alexandra Silva. Vol. 12759. Lecture Notes in Computer Science. virtual: Springer, 2021, pp. 1–14. DOI: 10.1007/978-3-030-81685-8_26 (cit. on pp. 2, 14).

[ANP16] Zaruhi Aslanyan, Flemming Nielson, and David Parker. "Quantitative Verification and Synthesis of Attack-Defence Scenarios". In: *CSF* (June 27–July 1, 2016). Lisbon, Portugal: IEEE Computer Society, 2016, pp. 105–119. DOI: 10.1109/CSF.2016.15 (cit. on p. 3).

[Ari+20] Jaime Arias, Carlos E. Budde, Wojciech Penczek, Laure Petrucci, Teofil Sidoruk, and Mariëlle Stoelinga. "Hackers vs. Security: Attack-Defence Trees as Asynchronous Multi-agent Systems". In: *ICFEM* (Mar. 1–3, 2021). Ed. by Shang-Wei Lin, Zhe Hou, and Brendan Mahoney. Vol. 12531. Lecture Notes in Computer Science. Singapore: Springer, 2020, pp. 3–19. DOI: 10.1007/978-3-030-63406-3_1 (cit. on pp. 3, 19, 20).

[Arn+13] Florian Arnold, Axel Belinfante, Freark van der Berg, Dennis Guck, and Mariëlle Stoelinga. "DFTCalc: A Tool for Efficient Fault Tree Analysis". In: *SAFECOMP* (Sept. 24–27, 2013). Ed. by Friedemann Bitsch, Jérémie Guiochet, and Mohamed Kaâniche. Vol. 8153. Lecture Notes in Computer Science. Toulouse, France: Springer, Sept. 2013, pp. 293–301. DOI: 10.1007/978-3-642-40793-2_27 (cit. on p. 3).

[Arn+15] Florian Arnold, Dennis Guck, Rajesh Kumar, and Mariëlle Stoelinga. "Sequential and Parallel Attack Tree Modelling". In: *SAFECOMP Workshops* (Sept. 22, 2015). Ed. by Floor Koornneef and Coen van Gulijk. Vol. 9338. Lecture Notes in Computer Science. Delft, The Netherlands: Springer, 2015, pp. 291–299. DOI: 10.1007/978-3-319-24249-1_25 (cit. on p. 3).

[Bag+12] Alessandra Bagnato, Barbara Kordy, Per Håkon Meland, and Patrick Schweitzer. "Attribute Decoration of Attack-Defense Trees". In: *International Journal of Secure Software Engineering* 3.2 (2012), pp. 1–35. DOI: 10.4018/jsse.2012040101 (cit. on p. 2).

[Beh+01] Gerd Behrmann, Ansgar Fehnker, Thomas Hune, Kim Guldstrand Larsen, Paul Pettersson, Judi Romijn, and Frits W. Vaandrager. "Minimum-Cost Reachability for Priced Timed Automata". In: *HSCC* (Mar. 28–30, 2001). Ed. by Maria Domenica Di Benedetto and Alberto L. Sangiovanni-Vincentelli. Vol. 2034. Lecture Notes in Computer Science. Rome, Italy: Springer, 2001, pp. 147–161. ISBN: 3-540-41866-0. DOI: 10.1007/3-540-45351-2_15 (cit. on pp. 2, 7).

[Boz+19] Marco Bozzano, Harold Bruintjes, Alessandro Cimatti, Joost-Pieter Katoen, Thomas Noll, and Stefano Tonetta. "COMPASS 3.0". In: *TACAS, Part I* (Apr. 6–11, 2019). Ed. by Tomás Vojnar and Lijun Zhang. Vol. 11427. Lecture Notes in Computer Science. Prague, Czech Republic: Springer, Apr. 2019, pp. 379–385. DOI: 10.1007/978-3-030-17462-0_25 (cit. on p. 3).

[Che+09] Rémy Chevallier, Emmanuelle Encrenaz-Tiphène, Laurent Fribourg, and Weiwen Xu. "Timed Verification of the Generic Architecture of a Memory Circuit Using Parametric Timed Automata". In: *Formal Methods in System Design* 34.1 (Feb. 2009), pp. 59–81. DOI: 10.1007/s10703-008-0061-x (cit. on p. 3).

[Dal+06] George C. Dalton, Robert Mills, John Colombi, and Richard A. Raines. "Analyzing Attack Trees using Generalized Stochastic Petri Nets". In: *IEEE Information Assurance Workshop* (June 21–23, 2006). West Point, NY, USA, July 2006, pp. 116–123. DOI: 10.1109/IAW.2006.1652085 (cit. on p. 3).

[Dav+11] Alexandre David, Kim Guldstrand Larsen, Axel Legay, Marius Mikucionis, Danny Bøgsted Poulsen, Jonas van Vliet, and Zheng Wang. "Statistical Model Checking for Networks of Priced Timed Automata". In: *FORMATS* (Sept. 21–23, 2011). Ed. by Uli Fahrenberg and Stavros Tripakis. Vol. 6919. Lecture Notes in Computer Science. Aalborg, Denmark: Springer, 2011, pp. 80–96. DOI: 10.1007/978-3-642-24310-3_7 (cit. on p. 3).

[Dav+15] Alexandre David, Kim Guldstrand Larsen, Axel Legay, Marius Mikucionis, and Danny Bøgsted Poulsen. "Uppaal SMC tutorial". In: *International Journal on Software Tools for Technology Transfer* 17.4 (2015), pp. 397–415. DOI: 10.1007/s10009-014-0361-y (cit. on p. 3).

[FMC09] Igor Nai Fovino, Marcelo Masera, and Alessio De Cian. "Integrating cyber attacks within fault trees". In: *Reliability Engineering & System Safety* 94.9 (2009), pp. 1394–1402. DOI: 10.1016/j.ress.2009.02.020 (cit. on p. 2).

[Gad+16a] Olga Gadyatskaya, René Rydhof Hansen, Kim Guldstrand Larsen, Axel Legay, Mads Chr. Olesen, and Danny Bøgsted Poulsen. "Modelling Attack-defense Trees Using Timed Automata". In: *FORMATS* (Aug. 24–26, 2016). Ed. by Martin Fränzle and Nicolas Markey. Vol. 9884. Lecture Notes in Computer Science. Springer, 2016, pp. 35–50. DOI: 10.1007/978-3-319-44878-7_3 (cit. on pp. 3, 19).

[Gad+16b] Olga Gadyatskaya, Ravi Jhawar, Piotr Kordy, Karim Lounis, Sjouke Mauw, and Rolando Trujillo-Rasua. "Attack Trees for Practical Security Assessment: Ranking of Attack Scenarios with ADTool 2.0". In: *QEST* (Aug. 23–25, 2016). Ed. by Gul Agha and Benny Van Houdt. Vol. 9826. Lecture Notes in Computer Science. Québec City, QC, Canada: Springer, Aug. 2016, pp. 159–162. DOI: 10.1007/978-3-319-43425-4_10 (cit. on p. 3).

[GIM15] Marco Gribaudo, Mauro Iacono, and Stefano Marrone. "Exploiting Bayesian Networks for the Analysis of Combined Attack Trees". In: *Electronic Notes in Theoretical Computer Science* 310 (2015), pp. 91–111. DOI: 10.1016/j.entcs.2014.12.014 (cit. on p. 3).

[Her+16] Holger Hermanns, Julia Krämer, Jan Krcál, and Mariëlle Stoelinga. "The Value of Attack-Defence Diagrams". In: *POST* (Apr. 2–8, 2016). Ed. by Frank Piessens and Luca Viganò. Vol. 9635. Lecture Notes in Computer Science. Springer, 2016, pp. 163–185. DOI: 10.1007/978-3-662-49635-0_9 (cit. on pp. 3–5, 19).

[HV06] Martijn Hendriks and Marcel Verhoef. "Timed automata based analysis of embedded system architectures". In: *IPDPS* (Apr. 25–29, 2006). Rhodes Island, Greece: IEEE, 2006. DOI: 10.1109/IPDPS.2006.1639422 (cit. on p. 3).

[JLR15] Aleksandra Jovanović, Didier Lime, and Olivier H. Roux. "Integer Parameter Synthesis for Real-Time Systems". In: *IEEE Transactions on Software Engineering* 41.5 (2015), pp. 445–461. DOI: 10.1109/TSE.2014.2357445 (cit. on p. 14).

[KGS15] Rajesh Kumar, Dennis Guck, and Mariëlle Stoelinga. "Time Dependent Analysis with Dynamic Counter Measure Trees". In: *QAPL* (Apr. 11–12, 2015). France: INRIA, Apr. 2015 (cit. on p. 3).

[Kor+10] Barbara Kordy, Sjouke Mauw, Sasa Radomirovic, and Patrick Schweitzer. "Foundations of Attack-Defense Trees". In: *FAST* (Sept. 16–17, 2010). Ed. by Pierpaolo Degano, Sandro Etalle, and Joshua D. Guttman. Vol. 6561. Lecture Notes in Computer Science. Pisa, Italy: Springer, 2010, pp. 80–95. DOI: 10.1007/978-3-642-19751-2_6 (cit. on p. 2).

[Kor+14] Barbara Kordy, Sjouke Mauw, Sasa Radomirovic, and Patrick Schweitzer. "Attack-defense trees". In: *Journal of Logic and Computation* 24.1 (2014), pp. 55–87. DOI: 10.1093/logcom/exs029 (cit. on pp. 2, 3, 19).

[KPS14] Barbara Kordy, Ludovic Piètre-Cambacédès, and Patrick Schweitzer. "DAG-based attack and defense modeling: Don't miss the forest for the attack trees". In: *Computer Science Review* 13-14 (2014), pp. 1–38. DOI: 10.1016/j.cosrev.2014.07.001 (cit. on p. 2).

[KRS15] Rajesh Kumar, Enno Ruijters, and Mariëlle Stoelinga. "Quantitative Attack Tree Analysis via Priced Timed Automata". In: *FORMATS* (Sept. 2–4, 2015). Ed. by Sriram Sankaranarayanan and Enrico Vicario. Vol. 9268. Lecture Notes in Computer Science. Madrid, Spain: Springer, 2015, pp. 156–171. DOI: 10.1007/978-3-319-22975-1_11 (cit. on pp. 3, 10).

[KS17] Rajesh Kumar and Mariëlle Stoelinga. "Quantitative Security and Safety Analysis with Attack-Fault Trees". In: *HASE* (Jan. 12–14, 2017). Singapore: IEEE, 2017, pp. 25–32. DOI: 10.1109/HASE.2017.12 (cit. on pp. 2–4, 10, 14).

[Kum+18] Rajesh Kumar, Stefano Schivo, Enno Ruijters, Bugra Mehmet Yildiz, David Huistra, Jacco Brandt, Arend Rensink, and Mariëlle Stoelinga. "Effective Analysis of Attack Trees: A Model-Driven Approach". In: *FASE* (Apr. 14–20, 2018). Ed. by Alessandra Russo and Andy Schürr. Vol. 10802. Lecture Notes in Computer Science. Springer, 2018, pp. 56–73. DOI: 10.1007/978-3-319-89363-1_4 (cit. on pp. 2–4).

[KW18] Barbara Kordy and Wojciech Widel. "On Quantitative Analysis of Attack-Defense Trees with Repeated Labels". In: *POST* (Apr. 14–20, 2018). Ed. by Lujo Bauer and Ralf Küsters. Vol. 10804. Lecture Notes in Computer Science. Thessaloniki, Greece: Springer, 2018, pp. 325–346. DOI: 10.1007/978-3-319-89722-6_14 (cit. on p. 3).

[MSR06] Peter Mell, Karen Scarfone, and Sasha Romanosky. "Common Vulnerability Scoring System". In: *IEEE Security & Privacy* 4.6 (2006), pp. 85–89. DOI: 10.1109/MSP.2006.145 (cit. on p. 2).

[ODK99] Rodolphe Ortalo, Yves Deswarte, and Mohamed Kaâniche. "Experimenting with Quantitative Evaluation Tools for Monitoring Operational Security". In: *IEEE Transactions on Software Engineering* 25.5 (1999), pp. 633–650. DOI: 10.1109/32.815323 (cit. on p. 2).

[Pet+19] Laure Petrucci, Michał Knapik, Wojciech Penczek, and Teofil Sidoruk. "Squeezing State Spaces of (Attack-Defence) Trees". In: *ICECCS* (Nov. 10–13, 2019). Ed. by Jun Pang and Jing Sun. Guangzhou, China: IEEE, 2019, pp. 71–80. DOI: 10.1109/ICECCS.2019.00015 (cit. on p. 3).

[Ros+08] Louis M. Rose, Richard F. Paige, Dimitrios S. Kolovos, and Fiona Polack. "The Epsilon Generation Language". In: *ECMDA-FA* (June 9–13, 2008). Ed. by Ina Schieferdecker and Alan Hartman. Vol. 5095. Lecture Notes in Computer Science. Berlin, Germany: Springer, 2008, pp. 1–16. DOI: 10.1007/978-3-540-69100-6_1 (cit. on p. 15).

[RR08] NATO Research and Technology Organisation (RTO). *Improving Common Security Risk Analysis*. Tech. rep. AC/323(ISP-049)TP/193. North Atlantic Treaty Organisation, University of California, Berkeley, 2008 (cit. on p. 2).

[RS14] Anne Remke and Mariëlle Stoelinga, eds. *Proceedings of the International Autumn School on Stochastic Model Checking (ROCKS 2012)* (Oct. 22–26, 2012). Vol. 8453. Lecture Notes in Computer Science. Vahrn, Italy: Springer, 2014. ISBN: 978-3-662-45488-6. DOI: 10.1007/978-3-662-45489-3 (cit. on p. 3).

[RS15] Enno Ruijters and Mariëlle Stoelinga. "Fault tree analysis: A survey of the state-of-the-art in modeling, analysis and tools". In: *Computer Science Review* 15 (2015), pp. 29–62. DOI: 10.1016/j.cosrev.2015.03.001 (cit. on pp. 2, 4).

[Sal+98] Chris Salter, O. Sami Saydjari, Bruce Schneier, and Jim Wallner. "Toward a secure system engineering methodology". In: *NSPW*. Charlottesville, Virginia, USA, Sept. 1998, pp. 2–10. DOI: 10.1145/310889.310900 (cit. on p. 2).

[Sch+17] Stefano Schivo, Bugra M. Yildiz, Enno Ruijters, Christopher Gerking, Rajesh Kumar, Stefan Dziwok, Arend Rensink, and Mariëlle Stoelinga. "How to Efficiently Build a Front-End Tool for UPPAAL: A Model-Driven Approach". In: *SETTA* (Oct. 23–25, 2017). Ed. by Kim Guldstrand Larsen, Oleg Sokolsky, and Ji Wang. Vol. 10606. Lecture Notes in Computer Science. Changsha, China: Springer, 2017, pp. 319–336. DOI: 10.1007/978-3-319-69483-2_19 (cit. on pp. 3, 5, 9, 14, 16, 21).

[Val+20] Samaikya Valluripally, Aniket Gulhane, Reshmi Mitra, Khaza Anuarul Hoque, and Prasad Calyam. "Attack Trees for Security and Privacy in Social Virtual Reality Learning Environments". In: *CCNC* (Jan. 10–13, 2020). IEEE, 2020, pp. 1–9. DOI: 10.1109/CCNC46108.2020.9045724 (cit. on p. 3).

[Wan00] Farn Wang. "Parametric Analysis of Computer Systems". In: *Formal Methods in System Design* 17.1 (2000), pp. 39–60. DOI: 10.1023/A:1008782501688 (cit. on p. 7).

[Wid+19] Wojciech Widel, Maxime Audinot, Barbara Fila, and Sophie Pinchinat. "Beyond 2014: Formal Methods for Attack Tree-based Security Modeling". In: *ACM Computing Surveys* 52.4 (2019), 75:1–75:36. DOI: 10.1145/3331524 (cit. on p. 3).

[WSJ07] Lingyu Wang, Anoop Singhal, and Sushil Jajodia. "Toward measuring network security using attack graphs". In: *QoP* (Oct. 29, 2007). Ed. by Günter Karjoth and Ketil Stølen. Alexandria, VA, USA: ACM, Oct. 2007, pp. 49–54. DOI: 10.1145/1314257.1314273 (cit. on p. 2).