

Attack time analysis in dynamic attack trees via integer linear programming

Milan Lopuhaä-Zwakenberg
University of Twente
The Netherlands
m.a.lopuhaa@utwente.nl

Mariëlle Stoelinga
University of Twente & Radboud University
The Netherlands
m.i.a.stoelinga@utwente.nl

Abstract—Attack trees are an important tool in security analysis, and an important part of attack tree analysis is computing metrics. This paper focuses on dynamic attack trees and their *min time* metric. For general attack trees, calculating *min time* efficiently is an open problem, with the fastest current method being enumerating all minimal attacks, which is NP-hard. This paper introduces 3 new tools for calculating *min time*. First, we show that static attack trees can be handled by a fast bottom-up algorithm. Second, we introduce a novel method for general dynamic attack trees based on mixed integer linear programming. Third, we show how the computation can be sped up by identifying the modules of an attack tree, i.e. subtrees connected to the rest of the attack tree via only one node. Experiments on a generated testing set of large attack trees verify that these methods have a large impact on performance.

Index Terms—Attack trees, quantitative analysis, optimization, mixed integer linear programming.

I. INTRODUCTION

(Dynamic) attack trees. Attack trees are a prominent methodology in security analysis. They facilitate security specialists in identifying, documenting, analyzing and prioritizing cyber risks. Attack trees are included in several popular system engineering frameworks, e.g. *UMLsec* [1] and *SysMLsec* [2], and are supported by industrial tools such as Isograph’s *AttackTree* [3].

An attack tree is an hierarchical diagram that describes a system’s vulnerabilities to an adversary’s attacks. Attack trees have been used in many different scenarios, such as IoT insider threats [4], electronic voting [5], and military information infrastructure [6]. Their popularity is owed to their simplicity on one hand, which allows



Fig. 1: The nodes of a dynamic attack tree.

for a wide range of applications, and their analyzability on the other hand. Despite their name, attack trees are rooted directed acyclic graphs. Its root represents the adversary’s goal, while the leaves represent basic attack steps (BASes) undertaken by the adversary. Each internal root is labeled with a gate, determining how its activation depends on that of its children.

Standard attack trees (SATs, also called static attack trees) feature only OR and AND gates, but many extensions have been introduced to describe more elaborate attack scenarios [7]. One of the most prominent extensions are *dynamic attack trees* (DATs) [8]. DATs introduce a SAND (“sequential AND”) gate, which is activated only when its children are activated sequentially. By contrast, an AND-node’s children can be activated in parallel. An epic example is given in Figure 2.

Quantitative analysis. Quantitative analysis aims at computing *attack tree metrics*. Such metrics are key performance indicators that formalize how well a system performs in terms of security. These metrics are essential when comparing alternatives or making trade-offs. Many such metrics exist, such as the minimal cost, minimal required skill, or maximal damage of a successful attack. This paper focuses on the metric *min time*: the minimal time the adversary needs to perform a successful attack, given the duration of the BASes.

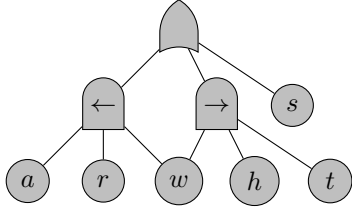


Fig. 2: A DAT for the Trojan War. To conquer Troy, the glancing-eyed Achaeans must either: collect wood (w), build a battering ram (r), and then assault Troy (a); collect wood, build a horse (h), and then trick the Trojans (t); or starve out the Trojans (s).

The metric *min time* is an important metric, since the success of a security attack crucially depends on time: attacks that take too long are not viable. Insight in timing behaviors of attacks is therefore a key to devising effective countermeasures. Note that *min time* is especially relevant in the context of DATs: On many metrics, such as cost, probability, skill required, the SAND and AND gates behave identical. Thus, to compute those metrics, the algorithms developed for SATs immediately generalize to DATs. It is in the timing behavior that the difference between SAND and AND manifests itself, so that novel computation algorithms are needed.

Algorithms for *min time*. The problem of computing *min time* is as follows: if each BAS is given a duration, what is the minimum time needed to reach the root of the attack tree?

The algorithms to compute *min time* crucially depend on two factors [9]: (1) the shape of the attack tree, i.e., trees vs DAGs and (2) the presence of SAND gates, i.e., SATs vs DATs.

Tree-shaped DATs. A (static or dynamic) attack tree that is tree-shaped can be computed via an intuitive bottom up algorithm, by propagating values from the leaves to the top. This algorithm works for general attributes (e.g. cost, probability, time, skill), by using appropriate operators to interpret how the values propagate through the gates. This bottom-up method was first proposed by [10] for static trees and later extended in [8] to dynamic attack trees. Their correctness of this algorithm crucially relies on subtle distributivity laws of the

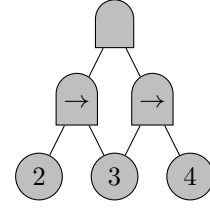


Fig. 3: Example DAT for which the bottom-up algorithm does not work.

chosen operators.

For DAG-shaped trees, the bottom up algorithm does not work in general, essentially because the values in different branches are no longer independent. For example, in the DAT of Figure 3 the bottom-up algorithm of [8] will calculate *min time* as $\max(2 + 3, 3 + 4) = 7$. However, the only successful attack is the one that activates the three BASes sequentially, and so *min time* equals $2 + 3 + 4 = 9$. For DAG-shaped SATs [9] proposes an efficient procedure exploiting *binary decision diagrams (BDDs)* that computes generic metrics over static DAGs. BDDs are compact representations of Boolean functions, and since static attack trees are Boolean functions, this is a natural fit. The algorithms in [9] work for a general class of metrics that take values in a so-called *attribute domain*, i.e. a triple (V, Δ, ∇) of a set V with two commutative associative operators Δ and ∇ , where ∇ distributes over Δ .

A key contribution of this paper is the realization that the bottom-up algorithm introduced in [8] for tree-like DATs can calculate *min time* for any *static* attack tree, even DAG-shaped. The key reason is that the attribute domain (\mathbb{R}, \max, \min) corresponding to *min time* is *idempotent*, i.e., $\min(x, x) = \max(x, x) = x$. This yields an enormous complexity reduction with respect to [9]: while the BDD-approach is exponential, the bottom-up method is linear in the size (number of nodes and edges) of the attack tree.

DAG-shaped DATs. In [9], efficient computation for DAG-shaped dynamic attack trees is left as an open problem. *min time* is computed by an enumerative method: by first generating all minimal attacks, and taking the one with the lowest attack time. Clearly this is computationally expensive.

In this paper, we present a novel method to

	Tree	DAG
Static	Bottom-up	BDDs → Bottom-up
Dynamic	Bottom-up	Enumerative → MILP

TABLE I: Algorithms for *min time*. We improve the static DAG-case from BDDs to bottom-up and for dynamic DAGs from enumerative to MILP.

calculate *min time* for general DATs based on *Mixed-Integer Linear Programming (MILP)*. Concretely, we translate calculating *min time* into an optimization problem. The natural formulation of *min time* as an optimization problem, however, yields a set of constraints that is nonlinear. We rewrite these into linear constraints by introducing auxiliary integer variables. Since dedicated solvers exist for MILP, this speeds up computation time considerably.

Modular analysis. To improve performance, we combine MILP with modular analysis [11]. Concretely, we identify independent *modules* in a DAT, i.e., rooted subDATs whose nodes are only connected to the rest of the DAT via the root of the subDAT. We show that modules can be analyzed separately. Further, if a module is tree-shaped or static, then we can deploy the more efficient bottom-up algorithm. We integrate these modules into our MILP algorithm.

Generalized semantics. Another point we settle in this paper is a generalized semantics for DATs. As SAND gates require their children to be executed consecutively, different branches in the DAT may impose conflicting restrictions on the execution orders. For example, the DAT $\text{SAND}(a, a)$ demands that a is executed before a , which is infeasible. To rule out these conflicts, [9] imposed well-formedness criteria. However, these also ruled out some satisfiable DATs. Furthermore, the definition of an attack in that work was overly restrictive, and for some DATs the fastest attack is not an attack under that definition. This leads to an overestimation of *min time*. In this work we extend the definition of a (successful) attack so *min time* is correctly defined. This new definition has the additional advantage of being applicable to all DATs, not just the well-formed ones.

Experimental validation. We compare the performance of four methods (modular versus nonmodular and enumerative versus MILP) on a set 750

DATs, obtained by combining smaller DATs from the literature. The experiments show that on larger DATs modular MILP has the best performance, followed by modular enumerative.

Contributions. Summarized our main contributions are:

- 1) A generalization of the poset semantics of [9] that significantly relaxes the syntactic constraints on the use of SAND-gates.
- 2) A novel algorithm to calculate *min time* for general DATs based on MILP.
- 3) A proof that bottom-up algorithm correctly calculates *min time* for all static ATs, including DAG-shaped ones.
- 4) A modularization approach that yields significant speed ups by separately handling fragments of the DAT that are static or tree-shaped.
- 5) Extensive experimental validation to evaluate the performance of the algorithms.

Data sources. The code for the experiments, the generated DATs and the experimental results are available in [12].

Paper organization. The structure of this paper is as follows. In Section II we discuss related literature. In Section III we review DATs and their semantics. In Section IV we show how MILP can be used to find *min time*. In Section V we discuss how modular analysis can cut down computation time, and in Section VI we perform experiments to compare MILP and modular analysis to the enumerative approach.

II. RELATED WORK

Dynamic attack trees were first formally defined in [8], although the addition of a sequential operator to attack trees had been discussed in earlier work [13], [14]. In [8] semantics for DATs are defined in terms of series-parallel graphs, based on the multiset semantics for standard attack trees in [10]. These semantics assume that each node must be activated separately for each of its parents. Effectively, this turns any DAG-like DAT into a tree-shaped one, which limits the range of scenarios that can be modeled.

Another approach to the semantics of DATs, using posets, is developed in [9]. In that paper each node can be activated only once, which allows for

a more flexible modeling of different scenarios. The paper also discusses how a general class of metrics on DATs can be computed, although the calculation of time-related metrics such as *min time* on general DATs is left as an open problem.

In [15] and [16] DATs are modeled as priced-timed automata. This allows for a detailed analysis, including the calculation of *min time*, but the interpretation of SAND-nodes is different from our paper: rather than imposing a sequential order on BAS activations, it determines in what order the children of a node are checked for being activated. This results in different semantics.

Closely related to DATs are *dynamic fault trees* (DFTs), which are used to model safety rather than security threats [17]. Temporal relations in DFTs are typically expressed by other means than SAND-nodes. The poset semantics of DFTs are studied via extended binary decision diagrams in [18].

III. DYNAMIC ATTACK TREES

This section reviews the definition of dynamic attack trees, and develops their semantics and the *min time* metric. The following definition of a DAT is from [9].

Definition 1. A dynamic attack tree (DAT) is a rooted directed acyclic graph $T = (N, E)$ where each node v has a type $\gamma(v) \in \{\text{BAS}, \text{OR}, \text{AND}, \text{SAND}\}$ such that $\gamma(v) = \text{BAS}$ if and only if v is a leaf, and every node v with $\gamma(v) = \text{SAND}$ has an ordering of its set of children.

Note that a DAT, despite its name, is not necessarily a tree. If it is, we call it *treelike*.

The root is denoted R_T . For $\gamma \in \{\text{BAS}, \text{OR}, \text{AND}, \text{SAND}\}$, we write N_γ for the set of nodes v with $\gamma(v) = \gamma$. If $\gamma(v) = \text{SAND}$ and v has (ordered) children v_1, \dots, v_n , we write $v = \text{SAND}(v_1, \dots, v_n)$ for convenience. We do the same for OR and AND, where the ordering of the children does not matter. We write T_v for the subDAG consisting of all *descendants* of v , i.e. all v' for which there is a path from v to v' . Furthermore, we let B_v be the set of descendants of v in N_{BAS} , i.e. the $a \in N_{\text{BAS}}$.

A dynamic attack tree codifies the ways an attacker can make a system fail by executing the *basic attack steps* i.e. the nodes in N_{BAS} . A non-BAS node is reached depending on its children,

where OR and AND have the expected meaning, and a SAND-node is reached if all its children are reached in their given order (the precise semantics are defined below). The adversary's goal is to reach the root node R_T .

In the literature, two interpretations of BASes with multiple parent nodes exist. The difference affects both the semantics and the metrics. In the first interpretation, *multiple activation* (MA), [10], [8], [19] each BAS can be activated multiple times, and every parent of a node requires its own activation of that node. In this interpretation $\text{SAND}(a, a)$ succeeds only if a is activated twice consecutively. By replacing nodes with multiple parents by one copy for each parent, any DAT can be transformed into a treelike one with equivalent semantics and metrics. As a result, quantitative analysis is 'solved' in the sense that metrics can be calculated quickly via a bottom-up algorithm [20]. The downside is that MA cannot adequately model systems in which one action can have multiple independent consequences.

In the other interpretation, *single activation* (SA), [9], [21], each BAS is executed at most once, and a node only needs to be activated once to count as an input for all its parent nodes. For example, in this interpretation $\text{SAND}(a, a)$ cannot be satisfied, because a cannot be activated before itself. SA has the advantage of being able to describe a much wider range of systems. However, the downside is that quantitative analysis is NP-hard in general [9].

For the remainder of this paper we follow SA in order to be able to model more realistic scenarios. Note that the two interpretations are the same for treelike DATs. Since every DAT is equivalent to a treelike one under MA, SA can model every scenario that MA can.

A. Semantics

In this section, we discuss the semantics of DATs. We mostly follow [9], though we need a more general definition because that work only defines the semantics of so-called *well-formed* DATs.

An attack (A, \prec) consists of a set A of BASes activated by the attacker, and a strict partial order \prec , where $a \prec a'$ means that a is executed before a' .

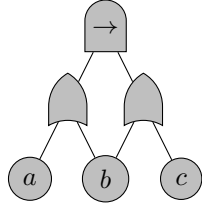


Fig. 4: An example of a DAT.

Definition 2. The set \mathcal{A}_T of attacks on T is defined as the set of strictly partially ordered sets $\mathcal{O} = (A, \prec)$, where $A \subseteq N_{\text{BAS}}$. This set has a partial order \leq given by $\mathcal{O} \leq \mathcal{O}'$, for $\mathcal{O} = (A, \prec)$ and $\mathcal{O}' = (A', \prec')$, if and only if $A \subseteq A'$ and $\prec \subseteq \prec'$.

We are interested in successful attacks, i.e. attacks that manage to reach the root node. Successful attacks are defined recursively as follows:

Definition 3. Let v be a node. We say that an attack $\mathcal{O} = (A, \preceq)$ reaches v if:

- 1) $v \in N_{\text{BAS}}$ and $v \in A$;
- 2) $v = \text{OR}(v_1, \dots, v_n)$ and \mathcal{O} reaches at least one of the v_i ;
- 3) $v = \text{AND}(v_1, \dots, v_n)$ and \mathcal{O} reaches all of the v_i ;
- 4) $v = \text{SAND}(v_1, \dots, v_n)$ and \mathcal{O} reaches all of the v_i , and for all $a_i \in \mathcal{O} \cap B_{v_i}$, $a_{i+1} \in \mathcal{O} \cap B_{v_{i+1}}$ one has $a_i \prec a_{i+1}$.

An attack is successful if it reaches R_T .

This allows us to define the semantics of T :

Definition 4. The semantics of a DAT T is the set \mathcal{S}_T of successful attacks on T .

The intuition behind a SAND-gate $v = \text{SAND}(v_1, \dots, v_n)$ is that it is only reached if all of the BASes of v_i have been (successfully) executed before any of the BASes of v_{i+1} has started.

We note that contrary to the static case (without SAND-gates), it is possible that $\mathcal{S}_T = \emptyset$. For example, $\mathcal{S}_{\text{SAND}(a,a)} = \emptyset$. Note that being successful is *not* monotonous on the set of attacks, i.e. it is possible that \mathcal{O} is successful while \mathcal{O}' is not, even if $\mathcal{O} \leq \mathcal{O}'$. For instance, in Figure 4 ($\{a, c\}, \{(a, c)\}$) is a successful attack, but ($\{a, b, c\}, \{(a, c)\}$) is not.

The key differences with [9] are as follows: In that work, attacks are called attacks only if they

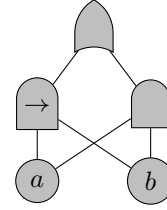


Fig. 5: An example of a well-formed DAT for which the semantics in [9] are insufficient.

satisfy the ordering constraints imposed by all SAND-gates. This is defined only for *well-formed* DATs, i.e., DATs for which all these ordering constraints put together are still satisfiable. More formally, that work only considers attacks that we call *full* in the following definition.

Definition 5. Let T be a DAT. Define a relation \sqsubseteq' on N_{BAS} by $a \sqsubseteq' a'$ if and only if there exists a node $v = \text{SAND}(v_1, \dots, v_n)$ and an $i < n$ such that $a \in B_{v_i}$ and $a' \in B_{v_{i+1}}$. Let \sqsubseteq be the transitive closure of \sqsubseteq' .

- 1) T is called well-formed if \sqsubseteq is a strict partial order.
- 2) An attack (A, \prec) on a well-formed DAT is called full if \prec equals $\sqsubseteq \upharpoonright_A$.

However, not all attacks will be full, because an attack may not need to reach all SAND-nodes in order to reach the root, and non-reached nodes should not put restrictions on attacks. Consider, for instance, the well-formed DAT of Figure 5. Only $(\{a, b\}, \{(a, b)\})$ is a full successful attack. However, $(\{a, b\}, \emptyset)$ is a successful attack as well. We see that non-full attacks are needed to fully describe the semantics of well-formed DATs, which motivates our Definition 2. Furthermore, our definition defines the semantics of general DATs, not just the well-formed ones.

B. The min time metric

This paper focuses on calculating *min time*, i.e., the minimal time it takes to perform a successful attack on a given DAT. While other metrics exist for DATs, *min time* is a fundamental time metric, and calculating *min time* for non-treelike DATs is an open problem [9].

Min time is defined as follows: Every BAS a has a *duration* $d_a \in \mathbb{R}_{\geq 0}$, denoting the time it

takes to executes a . If $a \prec a'$, then the BAS a' can only be started once a has been completed, while a and a' can be activated in parallel if such a relation does not exist. As such, we can define the total duration of an attack as

$$t(\mathcal{O}, d) = \max_{\substack{C \\ \text{in } \mathcal{O} \\ \text{max. chain}}} \sum_{a \in C} d_a, \quad (1)$$

where the maximum is taken over the maximal chains (i.e., maximal linearly ordered subsets) of the strict poset \mathcal{O} . We will often omit d from the notation and write $t(\mathcal{O})$ if there is no confusion. Note that t is monotonous: if $\mathcal{O} \leq \mathcal{O}'$ one has $t(\mathcal{O}) \leq t(\mathcal{O}')$. We define *min time* as

$$\text{mt}(T, d) = \min_{\mathcal{O} \in \mathcal{S}_T} t(\mathcal{O}, d). \quad (2)$$

Again we omit the argument d when there is no confusion. Note that $\text{mt}(T) = \infty$ if $\mathcal{S}_T = \emptyset$. It follows from the monotonicity of t that the minimum in (2) is attained in one of the minimal elements of the poset \mathcal{S}_T . Hence when calculating $\text{mt}(T)$ it suffices to only look at the minimal elements of \mathcal{S}_T , as expressed by the following proposition.

Proposition 6. *Let $\mathcal{G}_T \subset \mathcal{S}_T$ such that \mathcal{G}_T contains all minimal elements of the poset (\mathcal{S}_T, \leq) . Then*

$$\text{mt}(T, d) = \min_{\mathcal{O} \in \mathcal{G}_T} t(\mathcal{O}, d). \quad (3)$$

In Appendix A we discuss a bottom-up method to find such \mathcal{G}_T . In the absence of other approaches such as a bottom-up approach, one can use this proposition to calculate $\text{mt}(T)$ as in Algorithm 1, which first finds such a \mathcal{G}_T as in Theorem 15, and then calculates $\text{mt}(T)$ via (3). Algorithm 1 has exponential time complexity, and generally finding the set of minimal attacks is NP-complete [9]. This means that there is a need to find other approaches that do not rely on the set of minimal attacks.

Input: Dynamic attack tree T , duration vector $d \in \mathbb{R}^{N_{\text{BAS}}}$

Output: Min time $\text{mt}(T, d)$.

$\mathcal{G}_T \leftarrow$ as in Theorem 15;

return $\min_{\mathcal{O} \in \mathcal{G}_T} t(\mathcal{O}, d)$

Algorithm 1: MT-Enum for a DAT T .

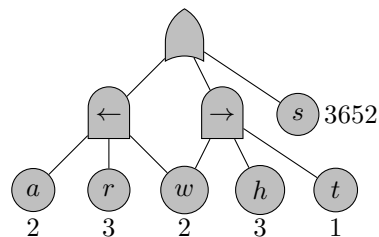
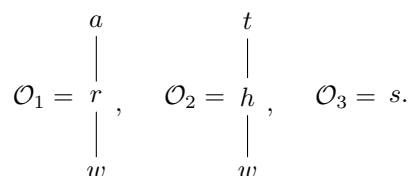


Fig. 6: The Trojan War DAT of Figure 2 augmented with durations.

Example 7. Figure 6 depicts the Trojan War DAT of Figure 2 augmented with durations for the BASes. To calculate $\text{mt}(T)$ we first find the minimal attacks. Represented as Hasse diagrams these are



These have duration $t(\mathcal{O}_1) = 2 + 3 + 2 = 7$, $t(\mathcal{O}_2) = 2 + 3 + 1 = 6$, and $t(\mathcal{O}_3) = 3652$. It follows that $\text{mt}(T) = \min\{7, 6, 3652\} = 6$.

IV. AN MILP APPROACH TO *min time*

This section describes a novel method to compute $\text{mt}(T)$ based on mixed-integer linear programming (MILP). Although MILP is NP-complete, a number of good heuristics and solvers exist specifically for MILP, which can result in a low computation time. We first show that *min time* can be found by solving an optimization problem in Theorem 9, and then we describe how that optimization problem can be rewritten into the MILP framework.

The building block of the new approach is the notion of time assignment, which assigns to each node a completion time f_v that respects all timing constraints in the DAT. If $f_v = \infty$ then v is not reached at all. The formal definition is stated below; recall that B_v is the set of BAS-descendants of v .

Definition 8. *Let T be a DAT. For a node v with children v_1, \dots, v_n and $i < n$, define*

$$Z_i^v := B_{v_i} \times B_{v_{i+1}}. \quad (4)$$

A time assignment is a vector $f \in [0, \infty]^N$ satisfying:

1) For each $a \in N_{\text{BAS}}$,

$$f_a \geq d_a; \quad (5)$$

2) For each $v = \text{OR}(v_1, \dots, v_n)$,

$$f_v \geq \min_i f_{v_i}; \quad (6)$$

3) For each $v = \text{AND}(v_1, \dots, v_n)$,

$$f_v \geq \max_i f_{v_i}; \quad (7)$$

4) For each $v = \text{SAND}(v_1, \dots, v_n)$, the following must hold:

a) it holds that

$$f_v \geq f_{v_n}; \quad (8)$$

b) If there is a $i \leq n$ such that $f_{v_i} = \infty$, then

$$f_v = \infty; \quad (9)$$

c) If there exist $i < n$ and $(a, a') \in Z_i^v$ such that $f_{a'} - d_{a'} < f_a < \infty$, then

$$f_v = \infty. \quad (10)$$

The set of all time assignments for T is denoted \mathcal{F}_T .

The conditions for SAND-nodes can be understood as follows. Equation (8) tells us that v cannot be reached before v_n , and (9) tells us that v cannot be reached if any of its children is not reached. Note that $f_{a'} - d_{a'}$ is the starting time of a BAS a' , and so (10) tells us that v can only be reached if the BASes of v_{i+1} are only started once those of v_i have been completed.

Note that we allow for a delay in completing node v , even when sufficiently many of its children have been completed. The following theorem relates time assignments to *min time*:

Theorem 9. $\text{mt}(T) = \min_{f \in \mathcal{F}_T} f_{R_T}$.

This theorem is proven in Appendix B. It allows us to calculate $\text{mt}(T)$ by solving the following optimization problem.

$$\begin{aligned} & \text{minimize}_{f \in [0, \infty]^N} f_{R_T} & (11) \\ & \text{s.t.} \quad (5) \text{ for all } a \in N_{\text{BAS}}, \\ & \quad (6) \text{ for all } v \in N_{\text{OR}}, \end{aligned}$$

$$(7) \text{ for all } v \in N_{\text{AND}},$$

$$(8) \text{--}(10) \text{ for all } v \in N_{\text{SAND}}.$$

Note that (11) is not an integral problem, due to the nonlinear constraints (6)–(10). We use auxiliary integer variables to transform these constraints into linear ones.

First, we need to get rid of the ∞ in equations (5)–(10), which we do by replacing it with a suitably large real number. Define the constant

$$M = 1 + \sum_{a \in N_{\text{BAS}}} d_a. \quad (12)$$

The following lemma shows that if T is satisfiable, then to minimize (11) one can just focus on the f that have $f_v \in [0, M-1] \cup \infty$. The lemma is proven in Appendix B.

Lemma 10. *There is a time assignment f minimizing (11) such that $f_v \in [0, M-1] \cup \infty$ for all v .*

This lemma shows that we can use M to play the role of ∞ where necessary. We enforce this by demanding $f_v \in [0, M]$, and we interpret $f_v = M$ to mean that v is not reached. For a node v , let n_v be its number of children, which are denoted v_1, \dots, v_{n_v} . We then use standard MILP techniques [22] to rewrite (6)–(10).

To rewrite (6), we introduce an auxiliary binary variable x_i^v for each $v \in N_{\text{OR}}$ and each $i \leq n_v$. The purpose of x_i^v is to represent the truthfulness of the statement “ $i = \arg \min_{i'} f_{v_{i'}}$ ”. We can then represent (6) by

$$\forall i \leq n_v: f_v \geq f_{v_i} + M(x_i^v - 1), \quad (13)$$

$$\sum_{i \leq n_v} x_i^v \geq 1. \quad (14)$$

Thus, Equation (13) is automatically satisfied if $x_i^v = 0$, and reduces to $f_v \geq f_{v_i}$ if $x_i^v = 1$. Equation (14) ensures that the latter must happen for at least one i , so together these encode $f_v \geq \min_i f_{v_i}$.

Equation (7) can be rewritten without auxiliary variables into

$$\forall i \leq n_v: f_v \geq f_{v_i}. \quad (15)$$

Finally, we consider (8)–(10). For $v \in N_{\text{SAND}}$, we introduce an auxiliary binary variable y^v that

$$\begin{array}{ll}
\text{minimize } f_{RT} & \\
\text{subject to} & \\
f_v \in \mathbb{R}, & \forall v \in N \\
f_a \geq d_a, & \forall a \in N_{\text{BAS}} \\
x_i^v \in \{0, 1\}, & \forall v \in N_{\text{OR}} \forall i \leq n_v \\
f_v \geq f_{v_i} + M(x_i^v - 1), & \forall v \in N_{\text{OR}} \forall i \leq n_v \\
\sum_{i \leq n_v} x_i^v \geq 1, & \forall v \in N_{\text{OR}} \\
f_v \geq f_{v_i}, & \forall v \in N_{\text{AND}} \forall i \leq n_v \\
y^v \in \{0, 1\}, & \forall v \in N_{\text{SAND}} \\
z_{i,a,a'}^v \in \{0, 1\}, & \forall v \in N_{\text{SAND}} \forall i < n_v \forall (a, a') \in Z_i^v \\
f_v \geq f_{v_{n_v}}, & \forall v \in N_{\text{SAND}} \\
f_v \geq M y^v, & \forall v \in N_{\text{SAND}} \\
y^v \geq \frac{1+f_{v_i}-M}{M}, & \forall v \in N_{\text{SAND}} \forall i < n_v \\
y^v \geq \frac{f_a-f_{a'}+d_{a'}}{M} - z_{i,a,a'}^v, & \forall v \in N_{\text{SAND}} \forall i < n_v \forall (a, a') \in Z_i^v \\
y^v \geq \frac{M-f_a+1}{M} + (z_{i,a,a'}^v - 1). & \forall v \in N_{\text{SAND}} \forall i < n_v \forall (a, a') \in Z_i^v
\end{array}$$

Fig. 7: Problem (11) rewritten in the MILP framework.

encodes “ $\exists i < n: f_{v_i} = \infty$ or $\exists i \exists (a, a') \in Z_i^v: f_{a'} - d_{a'} < f_a < \infty$.” Then we can write (8)–(10) as

$$f_v \geq f_{n_v}, \quad (16)$$

$$f_v \geq M y^v. \quad (17)$$

To ensure $y^v = 1$ whenever one of the $f_{v_i} = \infty$, we add the constraint

$$\forall i < n_v: y^v \geq \frac{1 + f_{v_i} - M}{M}, \quad (18)$$

which forces $y^v = 1$ only when $f_{v_i} > M - 1$. Furthermore, to ensure $y_v = 1$ whenever some a, a' satisfy $f_{a'} - d_{a'} < f_a$, we would like to add the constraint

$$\begin{array}{l}
\forall i < n_v \forall (a, a') \in Z_i^v: \\
y_v \geq \min \left\{ \frac{f_a - f_{a'} + d_{a'}}{M}, \frac{M - f_a}{M} \right\}. \quad (19)
\end{array}$$

This forces $y_v = 1$ only when both $f_{a'} - d_{a'} < f_a$ and $f_a < M$. To get rid of the minimum, we introduce an auxiliary variable $z_{i,a,a'}^v$ for each $i <$

n_v and $(a, a') \in Z_i^v$ as we did in (13) and (14). We then replace (19) with

$$\forall i < n_v \forall (a, a') \in Z_i^v: y_v \geq \frac{f_a - f_{a'} + d_{a'}}{M} - z_{i,a,a'}^v, \quad (20)$$

$$\forall i < n_v \forall (a, a') \in Z_i^v: y_v \geq \frac{M - f_a}{M} - (1 - z_{i,a,a'}^v). \quad (21)$$

Taking all of this together, it can be shown that the constraint $f_v \in [0, M]$ holds automatically for all ‘reasonable’ f and can be replaced by $f_v \in \mathbb{R}$. We then find that the optimization problem (11) can be rewritten into the MILP problem in Figure 7, and that *min time* can be found via Algorithm 2.

Input: Dynamic attack tree T , duration vector $d \in \mathbb{R}^{N_{\text{BAS}}}$

Output: Min time $\text{mt}(T, d)$.

$P \leftarrow$ problem of Figure 7;

return Solution of P

Algorithm 2: MT-MILP for a DAT T .

Note that the optimization of Figure 7 returns an f with $f_{RT} \leq M - 1$ if and only if $\mathcal{S}_T \neq \emptyset$. Hence

this optimization can also be used to determine whether T can successfully be attacked.

V. COMPUTATION TIME REDUCTION

In this section, we introduce an algorithm reducing the complexity of computing $\text{mt}(T)$. The algorithm consists of two components: First, we show that a bottom-up algorithm from [8] can be used to calculate *min time* for static (no SAND-nodes) and treelike DATs. As the state of the art method, based on binary decision diagrams [9], has exponential complexity, and the bottom-up algorithm has linear complexity, this is a big improvement. Second, we split up the calculation of *min time* into parts by identifying the *modules* of a DAT, i.e. subDAGs that are connected to the rest of the DAT via only one node.

A. Bottom-up computation

An important tool is the algorithm MT-BU introduced in [8] presented in Algorithm 3. Recall that T_v is subDAG of v consisting of the descendants of v .

Input: Dynamic attack tree T , duration vector $d \in \mathbb{R}^{N_{\text{BAS}}}$

Output: Potential min time $\text{mt}(T, d)$.

```

if  $\gamma(v) = \text{BAS}$  then
  | return  $d_v$ 
else if  $\gamma(v) = \text{OR}$  then
  | return  $\min_{v' \in \text{ch}(v)} \text{MT-BU}(T_{v'}, d|_{B_{v'}})$ 
else if  $\gamma(v) = \text{AND}$  then
  | return  $\max_{v' \in \text{ch}(v)} \text{MT-BU}(T_{v'}, d|_{B_{v'}})$ 
else //  $\gamma(v) = \text{SAND}$ 
  | return  $\sum_{v' \in \text{ch}(v)} \text{MT-BU}(T_{v'}, d|_{B_{v'}})$ 

```

Algorithm 3: MT-BU for a DAT T .

This algorithm attempts to calculate $\text{mt}(T)$ by traversing T bottom-up, which only has linear time complexity and is significantly faster than Algorithm 2. However, MT-BU does not correctly calculate $\text{mt}(T)$ for general T . It is shown that it does so for treelike T :

Theorem 11. [9] *If T is a treelike DAT, then MT-BU correctly calculates $\text{mt}(T)$.*

We call a DAT T *static* if $N_{\text{SAND}} = \emptyset$. The quantitative analysis of static attack trees (SATs) is studied in [9] for metrics of a general type

called *attribute domains*. A bottom-up algorithm does not work for calculating general metrics on general SATs, because a node with multiple parent nodes may be counted multiple times. However, for *min time* the relevant operations are \min and \max , which are *idempotent*, i.e., $\min(x, x) = \max(x, x) = x$. This means that it does not matter that nodes occur multiple times in one formula, and it allows us to prove the following theorem in Appendix C.

Theorem 12. *Assume T is static. Then MT-BU calculates $\text{mt}(T)$.*

The fastest state-of-the-art method for calculating *min time* for SATs is the binary decision diagram (BDD) approach of [9], where the Boolean function of a SAT is converted to a BDD, and a bottom-up algorithm on the BDD then calculates *min time*. However, the BDD of a Boolean function can be of exponential size, which means the overall approach has exponential time complexity in worst-case scenarios. By contrast, Algorithm 3 has linear time complexity, which makes a huge difference for large SATs.

Note that idempotency is a crucial property for MT-BU to work; in general, computing an attribute domain metric on SATs is NP-hard [9].

B. Modular analysis

While Algorithm 3 can greatly reduce complexity, it only does so in two relatively rare cases. However, it is possible to also reduce complexity when T is only partially static and/or treelike. A well-established method in studying attack trees is to consider the *modules* of T :

Definition 13. [11] *A module is a node $v \in N \setminus N_{\text{BAS}}$ such that all paths from $T \setminus T_v$ to T_v pass through v .*

Intuitively, the subtree T_v generated by a module v is connected to the rest of the tree only through v . As the following theorem shows, this allows us to subdivide the problem of finding $\text{mt}(T)$ into two parts: first we calculate $\text{mt}(T_v)$, and then we replace v by a new BAS with duration $\text{mt}(T_v)$, and calculate *min time* of the resulting DAT.

Theorem 14. *Let T be a DAT, and let v be a module of T . Let T^v be the node obtained by*

removing v and replacing v itself with a new BAS \tilde{v} , and let d^v be a duration vector for T^v given by

$$d_a^v = \begin{cases} d_a, & \text{if } a \in N_{\text{BAS}} \setminus B_v, \\ \text{mt}(T_v, d|_{B_v}), & \text{if } a = \tilde{v}. \end{cases} \quad (22)$$

Then $\text{mt}(T, d) = \text{mt}(T^v, d^v)$.

This theorem is proven in Appendix D. It reduces complexity in two ways: First, we split the tree up into two parts whose total size is the same as the original tree. Since MILP is NP-hard, this can have a big impact on computation time. Furthermore, the smaller DAT T_v can be static or treelike, in which case we can use MT-BU.

Input: Dynamic attack tree T , duration vector $d \in \mathbb{R}^{N_{\text{BAS}}}$, Algorithm \mathcal{A} to calculate *min time*

Output: Min time $\text{mt}(T)$.

$\mathcal{V} \leftarrow \text{Module}(T)$;

while $\mathcal{V} \neq \emptyset$ **do**

 Choose $v \in \mathcal{V}$ minimal;

if T_v is static or treelike **then**

$d_0 \leftarrow \text{MT-BU}(T_v, d|_{B_v})$;

else

$d_0 \leftarrow \mathcal{A}(T_v, d|_{B_v})$;

$(T, d) \leftarrow (T^v, d^v)$;

return d_{R_T} // R_T is a BAS now

Algorithm 4: \mathcal{A}_{Mod} for a DAT T . The notation T^v and d^v is from Theorem 14.

The resulting algorithm is displayed in Algorithm 4. Here `Module` refers to an algorithm that finds the modules of T ; this can be done with linear time complexity [11]. Algorithm \mathcal{A}_{Mod} makes use of an algorithm \mathcal{A} that calculates *min time*. For this one can use MT-Enum or MT-MILP, or potentially any new algorithm that calculates *min time*. Note that when T is treelike, every inner node is a module, in which case \mathcal{A}_{Mod} becomes equivalent to MT-BU for any \mathcal{A} .

VI. EXPERIMENTS

This section compares the performance of our methods. Since computation time only becomes relevant for large inputs, we want to test the methods on a set of suitably large DATs. However, to our knowledge there is no established

Source	$ N $	Treelike	Durations
[16] Fig. 1	12	no	unknown
[16] Fig. 8	20	no	unknown
[16] Fig. 9	12	no	unknown
[15] Fig. 1	16	yes	unknown
[14] Fig. 3	8	yes	known
[14] Fig. 5	21	yes	known
[14] Fig. 7	25	yes	known
[25] Fig. 2	20	yes	unknown
[26] Fig. 1	15	yes	unknown

TABLE II: DATs from the literature used as building blocks. The trees from [25] and [26] are attack-defense trees; only the root component of the attack part was used for these trees.

benchmark suite of DATs, and many DATs from the literature only have ≤ 25 nodes. Therefore, we create a testing set of DATs by combining a set of DATs from the literature into larger ones. Then, we compare (1) the MILP method MT-MILP to the enumerative algorithm MT-Enum and (2) the effect of modular analysis on performance time. Our experiments show that the MILP algorithm combined with modularization outperforms the other algorithms on larger DATs. The enumerative algorithm with modularization is the second best performing.

All experiments are performed on a PC with an Intel Core i7-7700HQ 2.8GHz processor and 32GB memory. All algorithms are implemented in Matlab, and for MILP we use the YALMIP environment [23] to translate the optimization problem into the Mosek solver [24], a state-of-the-art optimizer that can handle MILP problems. The code and results are available in [12].

A. Generation of testing DATs

In order to create a set of testing DATs that are large enough for a meaningful performance comparison, we do the following. As building blocks, we use a selection of DATs from the literature, presented in Table II. For some of the duration of the BASes were random variables, and we took the expected value for the duration; otherwise we took a random duration from $[1, 10]$. To combine two DATs T_1, T_2 into a larger one, we introduce a new root node v with a random label, and add edges (v, R_{T_1}) and (v, R_{T_2}) . Then, we pick random BASes b_1 from T_1 and b_2 from T_2 and identify them (with a new random duration). The new DAT represents a system consisting of

	Monolithic		Modular	
	Enumerative	MILP	Enumerative	MILP
	MT-Enum	MT-MILP	MT-Enum _{Mod}	MT-MILP _{Mod}
max time (\mathcal{T}_{80})	203.48	12706	1.9844	4.8906
mean log time (\mathcal{T}_{80})	-0.4788	0.0101	-0.4137	-0.2528
max time (\mathcal{T}_{150})			3810.1	40.438
mean log time (\mathcal{T}_{150})			0.2974	0.1935

TABLE III: Summary of the results. All times are in seconds. \mathcal{T}_{80} contains 400 DATs with ≤ 100 nodes, and \mathcal{T}_{150} contains 750 DATs with ≤ 168 nodes.

two subsystems that share one BAS. This is not the only way to combine two DATs into a single larger one, but we choose this method to ensure non-treelike DATs, as for treelike DATs the bottom-up algorithm MT-BU suffices.

For a given integer n_{\min} , we combine DATs randomly drawn from Table II via the method above until $|N| \geq n_{\min}$. We do this 5 times for each $1 \leq n_{\min} \leq 80$, giving us a testing set \mathcal{T}_{80} of 400 DATs with $8 \leq |N| \leq 100$. In order to compare performance on larger DATs, we also generate trees for $81 \leq n_{\min} \leq 150$, which together with \mathcal{T}_{80} gives us a testing set \mathcal{T}_{150} of 750 trees with $8 \leq |N| \leq 168$.

B. Time comparisons

We measure the computation time of the four algorithms on the testing sets. For the modular methods we use the testing set \mathcal{T}_{150} ; for the monolithic methods we use the smaller testing set \mathcal{T}_{80} as these methods turn out to take considerably more time. The results are summarized in Table III. All methods have a computation time of < 10 s for more than 95% of the testing sets, but they differ mostly in their outliers. Therefore we choose to report the mean log rather than the mean, which would be heavily skewed towards the outliers.

Table III shows that on \mathcal{T}_{80} , the monolithic (i.e. nonmodular) enumerative method is the fastest on typical trees, the outliers are considerably larger than for the two modular approaches. Of the modular approaches, the enumerative method performs better on \mathcal{T}_{80} , but the MILP method performs better on \mathcal{T}_{150} , suggesting that modular MILP is the best method for large DATs. This is also reflected in Figure 8, where the mean log time is given for the four algorithms, for DATs grouped by $|N|$. Furthermore, we see that the MILP approaches start off as considerably slower than their

enumerative counterparts, but scale better as $|N|$ increases.

More detailed one-on-one comparisons between algorithms are given in Figure 9. For most DATs, computation time is low, and the difference between the two methods is low. The difference between methods lies in the outliers. When comparing modular to monolithic approaches (b,c) only the monolithic approach has large outliers, showing the advantage of the modular approach. In (a) both algorithms have outliers, with those of MILP being considerably larger; the opposite is true in (b), where the outliers of the enumerative method are larger. Overall, these plots show that the modular MILP approach performs best, especially for outlier DATs.

VII. CONCLUSION AND DISCUSSION

This paper introduced two novel tools to calculate *min time* for DATs. First, we introduced a novel MILP-based approach that finds *min time* by phrasing it as an optimization problem. Second, we show how modular analysis can be used to reduce the computation time of any *min time* calculation algorithm. In the experiments, we compared these to the enumerative method, both monolithic and modular. The experiments show that for large DATs both MILP and modular analysis can have a big impact on computation time.

There are several directions in which this work can be expanded. First, it would be interesting to see whether an MILP approach can be used to calculate other metrics than just *min time*. Since MILP deals with real numbers, it will not work for general attribute domains in the sense of [9], but it is plausible that the MILP approach can be extended to other metrics, both on DATs and other types of attack trees.

Second, modular analysis can also be used for other metrics, as has been done for fault trees [27],

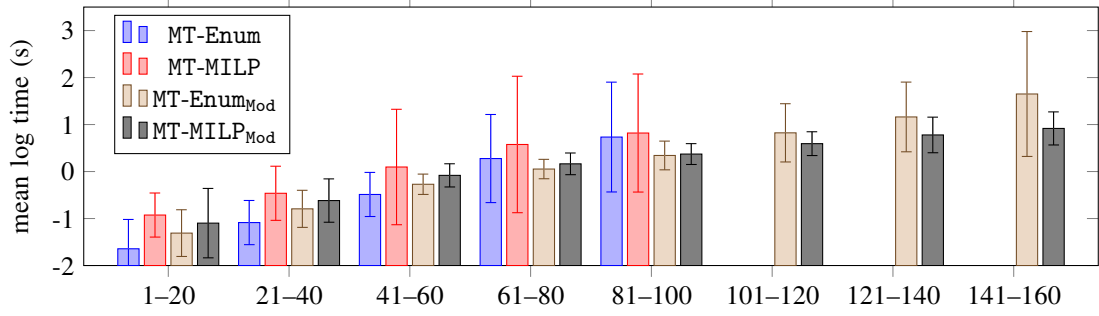


Fig. 8: Mean log time of the four DAT algorithms, grouped by the number of nodes $|N|$. Error bars denote standard deviation.

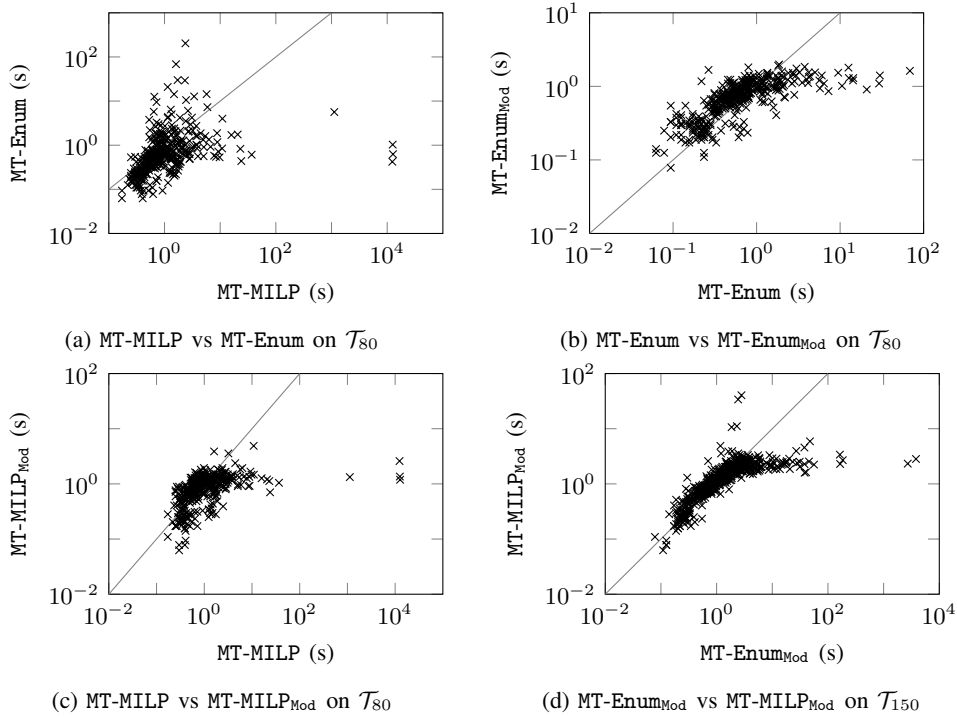


Fig. 9: Pairwise computation time comparisons of the four algorithms

[28]. Since modular analysis is a very general idea, a good approach would be to develop an axiomatization of metrics that can be handled via modular analysis, so that the method can be applied to a large set of metrics at once.

ACKNOWLEDGMENTS

This research has been partially funded by the ERC Consolidator grant 864075 CAESAR.

REFERENCES

- [1] Y. Roudier and L. Apvrille, "SysML-Sec: A model driven approach for designing safe and secure systems," in *MODELSWARD*. IEEE, 2015, pp. 655–664.
- [2] L. Apvrille and Y. Roudier, "SysML-sec: A sysML environment for the design and development of secure embedded systems," in *APCOSEC*, 2013. [Online]. Available: <http://www.eurecom.fr/publication/4186>
- [3] Isograph, *AttackTree*. [Online]. Available: <https://www.isograph.com/software/attacktree/>
- [4] F. Kammüller, J. R. Nurse, and C. W. Probst, "Attack tree analysis for insider threats on the iot using isabelle,"

- in *International Conference on Human Aspects of Information Security, Privacy, and Trust*. Springer, 2016, pp. 234–246.
- [5] A. Yasinisac and J. H. Pardue, “A process for assessing voting system risk using threat trees,” *Journal of Information Systems Applied Research*, vol. 4, no. 1, p. 4, 2011.
- [6] J. R. Surdu, J. M. Hill, R. Dodge, S. Lathrop, and C. Carver, “Military academy attack/defense network simulation,” in *Proceedings of advanced simulation technology symposium. Military, government, and aerospace simulation*, 2003.
- [7] B. Kordy, L. Piètre-Cambacédès, and P. Schweitzer, “Dag-based attack and defense modeling: Don’t miss the forest for the attack trees,” *Computer science review*, vol. 13, pp. 1–38, 2014.
- [8] R. Jhavar, B. Kordy, S. Mauw, S. Radomirović, and R. Trujillo-Rasua, “Attack trees with sequential conjunction,” in *IFIP International Information Security and Privacy Conference*. Springer, 2015, pp. 339–353.
- [9] C. E. Budde and M. Stoelinga, “Efficient algorithms for quantitative attack tree analysis,” pp. 1–15, 2021.
- [10] S. Mauw and M. Oostdijk, “Foundations of attack trees,” in *International Conference on Information Security and Cryptology*. Springer, 2005, pp. 186–198.
- [11] Y. Dutuit and A. Rauzy, “A linear-time algorithm to find modules of fault trees,” *IEEE Transactions on Reliability*, vol. 45, no. 3, pp. 422–425, 1996.
- [12] M. Lopuhaä-Zwakenberg, “Attack time analysis in dynamic attack trees via integer linear programming,” <https://figshare.com/s/d61b7f95c9d622f393e2>, DOI:10.4121/16685011.
- [13] S. A. Camepe and B. Yener, “Modeling and detection of complex attacks,” in *2007 Third International Conference on Security and Privacy in Communications Networks and the Workshops-SecureComm 2007*. IEEE, 2007, pp. 234–243.
- [14] F. Arnold, H. Hermanns, R. Pulungan, and M. Stoelinga, “Time-dependent analysis of attacks,” in *International Conference on Principles of Security and Trust*. Springer, 2014, pp. 285–305.
- [15] F. Arnold, D. Guck, R. Kumar, and M. Stoelinga, “Sequential and parallel attack tree modelling,” in *International Conference on Computer Safety, Reliability, and Security*. Springer, 2014, pp. 291–299.
- [16] R. Kumar, E. Ruijters, and M. Stoelinga, “Quantitative attack tree analysis via priced timed automata,” in *International Conference on Formal Modeling and Analysis of Timed Systems*. Springer, 2015, pp. 156–171.
- [17] G. Merle, “Algebraic modelling of dynamic fault trees, contribution to qualitative and quantitative analysis,” Ph.D. dissertation, École normale supérieure de Cachan-ENS Cachan, 2010.
- [18] W. Jiang, S. Zhou, L. Ye, D. Zhao, J. Tian, W. E. Wong, and J. Xiang, “An algebraic binary decision diagram for analysis of dynamic fault tree,” in *2018 5th International Conference on Dependable Systems and Their Applications (DSA)*. IEEE, 2018, pp. 44–51.
- [19] W. Wideł, M. Audinot, B. Fila, and S. Pinchinat, “Beyond 2014: Formal methods for attack tree-based security modeling,” *ACM Computing Surveys (CSUR)*, vol. 52, no. 4, pp. 1–36, 2019.
- [20] B. Fila and W. Wideł, “Exploiting attack–defense trees to find an optimal set of countermeasures,” in *2020 IEEE 33rd Computer Security Foundations Symposium (CSF)*. IEEE, 2020, pp. 395–410.
- [21] A. Jürgenson and J. Willemson, “Computing exact outcomes of multi-parameter attack trees,” in *OTM Confederated International Conferences “On the Move to Meaningful Internet Systems”*. Springer, 2008, pp. 1036–1051.
- [22] D.-S. Chen, R. G. Batson, and Y. Dang, *Applied integer programming: modeling and solution*. John Wiley & Sons, 2011.
- [23] J. Lofberg, “Yalmip: A toolbox for modeling and optimization in matlab,” in *2004 IEEE international conference on robotics and automation (IEEE Cat. No. 04CH37508)*. IEEE, 2004, pp. 284–289.
- [24] M. ApS, “Mosek optimization toolbox for matlab,” *User’s Guide and Reference Manual, Version*, vol. 4, 2019.
- [25] M. Fraile, M. Ford, O. Gadyatskaya, R. Kumar, M. Stoelinga, and R. Trujillo-Rasua, “Using attack-defense trees to analyze threats and countermeasures in an atm: a case study,” in *IFIP Working Conference on The Practice of Enterprise Modeling*. Springer, 2016, pp. 326–334.
- [26] B. Kordy and W. Wideł, “On quantitative analysis of attack–defense trees with repeated labels,” in *International Conference on Principles of Security and Trust*. Springer, 2018, pp. 325–346.
- [27] K. A. Reay and J. D. Andrews, “A fault tree analysis strategy using binary decision diagrams,” *Reliability engineering & system safety*, vol. 78, no. 1, pp. 45–56, 2002.
- [28] E. Ruijters and M. Stoelinga, “Fault tree analysis: A survey of the state-of-the-art in modeling, analysis and tools,” *Computer science review*, vol. 15, pp. 29–62, 2015.

APPENDIX A

SUFFICIENT SETS OF SUCCESSFUL ATTACKS

The aim of this appendix is to give a bottom-up algorithm that, for a DAT T , determines a set \mathcal{G}_T of successful attacks containing the minimal elements of \mathcal{S}_T . By Proposition 6 having such a \mathcal{G}_T can aid in calculating *min time*.

We find \mathcal{G}_T via a bottom-up procedure. We present the idea of the procedure in the following; the formal statement and proof is in Theorem 15. The idea is to find for every node v a set $g(v)$ of attacks reaching v , such that $g(v)$ contains all minimal attacks reaching v . We then take $\mathcal{G}_T = g(\mathcal{R}_T)$.

For a BAS a it suffices to take

$$g(a) = \{\{a\}, \emptyset\}, \quad (23)$$

as that set’s single element is the minimal attack reaching a . For a node $v = \text{OR}(v_1, v_2)$, an attack reaches v if it reaches either v_1 or v_2 . Therefore we take

$$g(v) = g(v_1) \cup g(v_2). \quad (24)$$

For $v = \text{AND}(v_1, v_2)$, an attack that reaches v has to reach both v_1 and v_2 . We get such attacks by taking attacks \mathcal{O} and \mathcal{O}' reaching v_1 and v_2 , respectively, and then taking the minimal attack containing both of them. Formally, this is done as follows. For a relation R on a set X , we denote its transitive closure by $\text{tr}(R)$. If $\mathcal{O} = (A, \prec)$ and $\mathcal{O}' = (A', \prec')$ are two attacks, then we define the attack $P(\mathcal{O}, \mathcal{O}') = (A_1, \prec_1)$ by

$$A_1 = A \cup A', \quad (25)$$

$$\prec_1 = \text{tr}(\prec \cup \prec'), \quad (26)$$

if \prec_1 is a strict partial order on A_1 ; otherwise we leave $P(\mathcal{O}, \mathcal{O}')$ undefined. Then $\mathcal{O}, \mathcal{O}' \leq P(\mathcal{O}, \mathcal{O}')$, and $P(\mathcal{O}, \mathcal{O}')$ is the minimal attack with that property if it exists, and no such attack exists if $P(\mathcal{O}, \mathcal{O}')$ does not exist. We then let $g(v)$ be the set of all $P(\mathcal{O}, \mathcal{O}')$, where $\mathcal{O} \in g(v_1)$ and $\mathcal{O}' \in g(v_2)$, for which $P(\mathcal{O}, \mathcal{O}')$ exists.

For $v = \text{SAND}(v_1, v_2)$ we do something similar, but we also need to add relations $a_1 \prec a_2$ for $a_1 \in B_{v_1}$ and $a_2 \in B_{v_2}$. Formally, for subsets $X, Y \subset N_{\text{BAS}}$, we define $L_{X,Y}(\mathcal{O}) = (A_2, \prec_2)$ by

$$A_2 = A, \quad (27)$$

$$\prec_2 = \text{tr}(\prec \cup ((A \cap X) \times (A \cap Y))), \quad (28)$$

if \prec_2 is a strict partial order on A_2 , and otherwise we leave it undefined. The intuition is that $S_{X,Y}(\mathcal{O})$ is equal to \mathcal{O} , except that it enforces that elements of X are activated before those of Y . Finally define $S_{X,Y}(\mathcal{O}, \mathcal{O}') = L_{X,Y}(P(\mathcal{O}, \mathcal{O}'))$; then $g(v)$ is the set of all $S_{B_{v_1}, B_{v_2}}(\mathcal{O}, \mathcal{O}')$, where $\mathcal{O} \in g(v_1)$ and $\mathcal{O}' \in g(v_2)$, for which $S_{B_{v_1}, B_{v_2}}(\mathcal{O}, \mathcal{O}')$ exists.

The following theorem formalizes the . For notational convenience, we only formulate it for binary T , but it can easily be expanded to the general setting.

Theorem 15. *Define a function $g: N \rightarrow \mathcal{P}(\mathcal{A}_T)$ by*

$$\forall a \in N_{\text{BAS}}: g(a) = \{\{\{a\}, \emptyset\}\}, \quad (29)$$

$$g(\text{OR}(v_1, v_2)) = g(v_1) \cup g(v_2), \quad (30)$$

$$g(\text{AND}(v_1, v_2)) = \left\{ X = P(\mathcal{O}_1, \mathcal{O}_2) : \begin{array}{l} \mathcal{O}_1 \in g(v_1), \\ \mathcal{O}_2 \in g(v_2), \\ X \text{ exists} \end{array} \right\}, \quad (31)$$

$$g(\text{SAND}(v_1, v_2)) = \left\{ Y = S_{B_{v_1}, B_{v_2}}(\mathcal{O}_1, \mathcal{O}_2) : \begin{array}{l} \mathcal{O}_1 \in g(v_1), \\ \mathcal{O}_2 \in g(v_2), \\ Y \text{ exists} \end{array} \right\}, \quad (32)$$

Then $\mathcal{G}_T := g(\mathcal{R}_T)$ is a set of successful attacks that contains the minimal elements of \mathcal{S}_T .

Proof. Let \mathcal{A}_v be the subset of \mathcal{A}_T of attacks activating v . We prove by induction that $g(v)$ is a subset of \mathcal{A}_v containing all minimal elements. For $\mathcal{O} \in \mathcal{A}_T$, we denote

$$U(\mathcal{O}) = \{\mathcal{O}' \in \mathcal{A}_T : \mathcal{O} \leq \mathcal{O}'\}. \quad (33)$$

We can distinguish the following cases:

- if $a \in N_{\text{BAS}}$, then an attack (A, \prec) activates a if and only if $a \in A$; hence $(\{a\}, \emptyset)$ is the unique minimal element of \mathcal{A}_v .
- if $v = \text{OR}(v_1, v_2)$, an attack \mathcal{O} is successful if $\mathcal{O} \in \mathcal{A}_{v_1} \cup \mathcal{A}_{v_2}$. By the induction hypothesis the set of such attacks that are minimal is a subset of $g(v_1) \cup g(v_2)$.
- if $v = \text{AND}(v_1, v_2)$, an attack \mathcal{O} is successful if $\mathcal{O} \in \mathcal{A}_{v_1} \cap \mathcal{A}_{v_2}$. Note that by induction we have

$$\mathcal{A}_{v_1} \cap \mathcal{A}_{v_2} = \bigcup_{\substack{\mathcal{O}_1 \in g(v_1), \\ \mathcal{O}_2 \in g(v_2)}} U(\mathcal{O}_1) \cap U(\mathcal{O}_2). \quad (34)$$

Take such $\mathcal{O}_1 = (A_1, \prec_1)$ and $\mathcal{O}_2 = (A_2, \prec_2)$. An attack $\mathcal{O} = (A, \prec)$ is an element of $U(\mathcal{O}_1) \cap U(\mathcal{O}_2)$ if and only if

$$A \supset A_1 \cup A_2, \quad (35)$$

$$\prec \supset \prec_1 \cup \prec_2. \quad (36)$$

Since \mathcal{O} is a strict poset, this means that $P(\mathcal{O}_1, \mathcal{O}_2) \leq \mathcal{O}$ if the former exists, and otherwise such an \mathcal{O} does not exist. It follows that

$$\mathcal{A}_{v_1} \cap \mathcal{A}_{v_2} = \bigcup_{\substack{\mathcal{O}_1 \in g(v_1), \mathcal{O}_2 \in g(v_2), \\ P(\mathcal{O}_1, \mathcal{O}_2) \text{ exists}}} U(P(\mathcal{O}_1, \mathcal{O}_2)), \quad (37)$$

hence $g(v)$ contains the minimal elements of \mathcal{A}_v .

- Suppose $v = \text{SAND}(v_1, v_2)$. If $\mathcal{O}_1 = (A_1, \prec_1)$ with $A_1 \cap B_{v_2} = \emptyset$ activates v_1 and \mathcal{O}_2 activates v_2 , then $S_{B_{v_1}, B_{v_2}}(\mathcal{O}_1, \mathcal{O}_2)$, if it exists, activates v . This shows that $g(v)$ contains

only attacks activating v . On the other hand, suppose that $\mathcal{O} = (A, \prec)$ activates v . Define

$$A_1 = A \cap B_{v_1}, \quad (38)$$

$$\prec_1 = \prec|_{A_1}, \quad (39)$$

$$A_2 = A \cap B_{v_2}, \quad (40)$$

$$\prec_2 = \prec|_{A_2}. \quad (41)$$

Then (A_1, \prec_1) activates v_1 , (A_2, \prec_2) activates v_2 . By the induction hypothesis, there exist $\mathcal{O}'_1 \leq (A_1, \prec_1)$ and $\mathcal{O}'_2 \leq (A_2, \prec_2)$ in $\mathcal{G}(v_1)$ and $\mathcal{G}(v_2)$, respectively; then

$$S_{B_{v_1}, B_{v_2}}(\mathcal{O}'_1, \mathcal{O}'_2) \leq \mathcal{O}. \quad (42)$$

This shows that $g(v)$ contains the minimal elements of \mathcal{A}_v . \square

APPENDIX B

PROOFS OF THEOREM 9 AND LEMMA 10

To prove Theorem 9 we need some auxiliary results. We start with a stricter definition of time assignments.

Definition 16. A time assignment f is called exact if equality holds in (6) and (7) for all v , and equality holds in (8) if the prerequisite conditions for (9) and (10) are not satisfied. The set of exact time assignments is denoted $\mathcal{F}_{E,T}$.

The advantage of exact time assignments is that they are easier to reason about. The following lemma shows that restricting ourselves to exact time assignments does not affect the minimum.

Lemma 17. $\min_{f \in \mathcal{F}_T} f_{R_T} = \min_{f \in \mathcal{F}_{E,T}} f_{R_T}$.

Proof. Let $f \in \mathcal{F}_T$, and define $f' \in \mathcal{F}_{E,T}$ by $f'_a = f_a$ for $a \in N_{\text{BAS}}$, and having f' satisfy equality in (6) and (7) for all v , and by satisfying equality in (8) if the prerequisite conditions for (9) and (10) are not satisfied. By induction it is straightforward to show that $f'_{R_T} \leq f_{R_T}$, which proves the lemma. \square

The following lemma has a straightforward proof by induction which is therefore omitted.

Lemma 18. Let f be an exact time assignment with $f_{R_T} < \infty$. Then $f_v \in \{f_a : a \in B_v\} \cup \{\infty\}$ for all $v \in N$. \square

We need two more lemmas as ingredients for the proof of Theorem 9.

Lemma 19. Let f be an exact time assignment. Then there exists a successful attack \mathcal{O} with $t(\mathcal{O}) \leq f_{R_T}$.

Proof. Let $C \in (0, \infty)$. Define the attack $\mathcal{O}_f^C = (A_f^C, \prec_f^C)$ by

$$A_f^C = \{a \in N_{\text{BAS}} : f_a \leq C\}, \quad (43)$$

$$\prec_f^C = \{(a, a') \in (A_f^C)^2 : f_a \leq f_{a'} - d_{a'}\}. \quad (44)$$

We prove by induction on T that for any f and C one has that \mathcal{O}_f^C reaches a node v when $f_v \leq C < \infty$. For convenience we assume that T is binary; this does not substantially alter the proof but makes the notation easier.

- Suppose $a \in N_{\text{BAS}}$, and suppose $f_a \leq C$. Then $a \in A_f^C$ and \mathcal{O}_f^C is successful.
- Suppose $v = \text{OR}(v_1, v_2)$ and $f_v \leq C < \infty$. Since $f_v = \min\{f_{v_1}, f_{v_2}\}$, we may assume WLOG that $f_{v_1} \leq C$. By the induction hypothesis the attack \mathcal{O}_f^C reaches v_1 . It follows that \mathcal{O}_f^C reaches v , and so it reaches v too.
- The case that $v = \text{AND}(v_1, v_2)$ is analogous to the OR-case.
- Suppose that $v = \text{SAND}(v_1, v_2)$ and $f_v \leq C < \infty$; then

$$f_{v_1} \leq f_{v_2} = f_v \leq C < \infty \quad (45)$$

and $f_a \leq f_{a'} - d_{a'}$ for all $a \in B_{v_1}$ and $a' \in B_{v_2}$ for those a with $f_a < \infty$. By the induction hypothesis, one proves as in the OR-case that \mathcal{O}_f^C reaches both v_1 and v_2 . Furthermore, by definition of \prec_f^C one has $a \prec_f^C a'$ for all $a \in A_f^C \cap B_{v_1}$ and $a' \in A_f^C \cap B_{v_2}$. We conclude that \mathcal{O}_f^C reaches v .

Now take $C = f_{R_T} < \infty$; the resulting attack $\mathcal{O}_f^{f_{R_T}}$ is successful. Lemma 18 then tells us that

$$f_{R_T} \in \{f_a : a \in A_f^{f_{R_T}}\}. \quad (46)$$

Since the maximum of the RHS is at most equal to f_{R_T} we find

$$f_{R_T} = \max_{a \in A_f^{f_{R_T}}} f_a. \quad (47)$$

On the other hand, by induction it can be shown straightforwardly that for every chain C in $\mathcal{O}_f^{f_{R_T}}$ one has

$$f_{\max C} \geq \sum_{a \in C} d_a. \quad (48)$$

Combining this with (47) we find $f_{R_T} \geq t(\mathcal{O}_f^{f_{R_T}})$. \square

Lemma 20. *Let \mathcal{O} be a successful attack. Then there exists an exact time assignment f such that $f_{R_T} \leq t(\mathcal{O})$.*

Proof. Let $\mathcal{O} = (A, \prec)$. Define an exact time assignment $f_{\mathcal{O}}$ as follows:

- If $a \in N_{\text{BAS}} \setminus A$, take $f_{\mathcal{O},a} = \infty$;
- If $a \in A$, define $f_{\mathcal{O}}$ recursively by

$$f_{\mathcal{O},a} = d_a + \max_{a' \prec a} f_{\mathcal{O},a'}, \quad (49)$$

where the maximum over the empty set equals 0;

- If $v \in N \setminus N_{\text{BAS}}$, define $f_{\mathcal{O},v}$ by taking equality in (6)–(8).

We prove by induction on T that $f_{\mathcal{O},v} < \infty$ if \mathcal{O} reaches v . For convenience we again assume that T is binary.

- When $v \in N_{\text{BAS}}$ it is clear from the definition.
- Suppose $v = \text{OR}(v_1, v_2)$. WLOG \mathcal{O} reaches v_1 , so by the induction hypothesis $f_{\mathcal{O},v_1} < \infty$. As $f_{\mathcal{O},v} \leq f_{\mathcal{O},v_1}$ the induction hypothesis holds for v .
- The case $v = \text{AND}(v_1, v_2)$ is analogous to the OR-case.
- Suppose $v = \text{SAND}(v_1, v_2)$. Since v is reached, this means that v_1 and v_2 are reached and that $a \prec a'$ for all $a \in B_{v_1} \cap A$, $a' \in B_{v_2} \cap A$. The first statement implies, by the induction hypothesis, that $f_{v_1}, f_{v_2} < \infty$. The second statement, together with the definition of $f_{\mathcal{O}}$ on BASes, implies that $f_a \leq f_{a'} - d_{a'}$ for a, a' as above, which means that the prerequisite condition for (10) never holds. It follows that $f_{\mathcal{O},v} < \infty$.

Since \mathcal{O} is successful one has $f_{\mathcal{O},R_T} < \infty$, so by Lemma 18 we know that

$$f_{\mathcal{O},R_T} \in \{f_a : a \in A\}. \quad (50)$$

However, by induction on the strict poset \mathcal{O} one can prove that $t(\mathcal{O}) = \max_{a \in A} f_{\mathcal{O},a}$. It follows that $t(\mathcal{O}) \geq f_{\mathcal{O},R_T}$, as was to be shown. \square

Proof of Theorem 9. By Lemma 17 it is enough to only consider exact time assignments. From Lemma 19 it follows that

$$\min_{\mathcal{O} \in \mathcal{S}_T} t(\mathcal{O}) \leq \min_{f \in \mathcal{F}_{E,T}} f_{R_T}. \quad (51)$$

On the other hand, Lemma 20 tells us that

$$\min_{\mathcal{O} \in \mathcal{S}_T} t(\mathcal{O}) \geq \min_{f \in \mathcal{F}_{E,T}} f_{R_T}. \quad (52)$$

Since the LHS of these two inequalities is equal to $\text{mt}(T)$, these two inequalities together prove the theorem. \square

Proof of Lemma 10. If $\mathcal{S}_T = \emptyset$, then the optimal f has $f_{R_T} = \infty$; hence the constant time assignment $f \equiv \infty$ also minimizes (11).

Suppose $\mathcal{S}_T \neq \emptyset$, and let f be a time assignment minimizing (11). Since $\mathcal{S}_T \neq \emptyset$ one has $f_{R_T} = \text{mt}(T) < \infty$. Let \mathcal{O} be as in Lemma 19, and let $f' = f_{\mathcal{O}}$ be as in the proof of Lemma 20. Then $f'_{R_T} \leq f_{R_T} = \text{mt}(T)$, but since f_{R_T} is minimal this is an equality. Furthermore, it is straightforward to prove by induction that $f'_v \leq M - 1$ for all v from the definition of $f_{\mathcal{O}}$. \square

APPENDIX C

PROOF OF THEOREM 12

If (A, \prec) is a successful attack, then so is (A, \emptyset) , as there are no SAND-gates to put restrictions on \prec . Furthermore,

$$\text{mt}(T) = \min_{(A, \emptyset) \in \mathcal{S}_T} \max_{a \in A} d_a. \quad (53)$$

We now prove the theorem by induction on v that $\text{MT-BU}(T_v, d|_{B_v}) = \text{mt}(T_v)$. If $v \in N_{\text{BAS}}$, then $(\{v\}, \emptyset)$ is the only successful attack on v , which indeed has duration $d_v = \text{MT-BU}(T_v, d|_{\{v\}})$. Now let $v = \text{OR}(v_1, \dots, v_n)$. A minimal successful attack on v is a minimal successful attack on one of the v_i . It follows that

$$\text{mt}(v) = \min_{(A, \emptyset) \in \mathcal{S}_{T_v}} \max_{a \in A} d_a \quad (54)$$

$$= \min_i \min_{(A, \emptyset) \in \mathcal{S}_{T_{v_i}}} \max_{a \in A} d_a \quad (55)$$

$$= \min_i \text{MT-BU}(T_{v_i}, d|_{B_{v_i}}) \quad (56)$$

$$= \text{MT-BU}(T_v, d|_{B_v}), \quad (57)$$

where the second equation follows from the induction hypothesis. This proves the theorem for v ; the AND-case is analogous.

APPENDIX D

PROOF OF THEOREM 14

To prove this theorem, we first prove two auxiliary lemmas. Throughout this section, we write $d_v := d|_{B_v}$ for convenience.

Lemma 21. *Let \mathcal{O} be a successful attack on T^v . then there is a successful attack \mathcal{O}' on T with $t(\mathcal{O}, d) = t(\mathcal{O}', d^v)$.*

Proof. Let $\mathcal{O} = (A, \prec)$. If $\tilde{v} \notin A$, then \mathcal{O} is also a successful attack on T , and $t(\mathcal{O}, d) = t(\mathcal{O}, d^v)$. Now suppose $\tilde{v} \in A$. Let $\mathcal{O}_- = (A_-, \prec_-)$ be a successful attack on T_v that satisfies $t(\mathcal{O}_-, d_v) = \text{mt}(T_v, d_v)$. Define an attack $\mathcal{O}' = (A', \prec')$ on T by $A' = A \setminus \{\tilde{v}\} \cup A_-$, and $a \prec' a'$ if and only if one of the following holds:

- $a, a' \in A \setminus \{\tilde{v}\}$ and $a \prec a'$;
- $a, a' \in A_-$ and $a \prec_- a'$;
- $a \in A \setminus \{\tilde{v}\}$, $a' \in A_-$, and $a \prec \tilde{v}$;
- $a \in A_-$, $a' \in A \setminus \{\tilde{v}\}$ and $\tilde{v} \prec a'$.

Intuitively, \mathcal{O}' is the order obtained by taking \mathcal{O} and replacing \tilde{v} with the attack \mathcal{O}_- . Since v is a module, $A \setminus \{\tilde{v}\}$ and A_- are disjoint, and \mathcal{O}' is an attack. Since it activates v and \mathcal{O} is a successful attack on T^v , the attack \mathcal{O}' is successful on T . Furthermore, the maximal chains C' of \mathcal{O}' are obtained by taking a maximal chain C of \mathcal{O} and replacing a possible instance of \tilde{v} with a maximal chain C_- of \mathcal{O}_- . From the definition of d_v^v it follows that if $\tilde{v} \notin C$ one has

$$\sum_{a \in C'} d_a = \sum_{a \in C} d_a^v, \quad (58)$$

and if $\tilde{v} \in C$ one has

$$\sum_{a \in C'} d_a = \sum_{a \in C} d_a^v + \sum_{a \in C_-} d_a - d_v^v \quad (59)$$

$$\geq \sum_{a \in C} d_a^v, \quad (60)$$

with equality if and only if C_- satisfies $\text{mt}(T_v, d_v) = \sum_{a \in C_-} d_a$. Taking the maximum over all C in both (58) and (60), we find that

$$t(\mathcal{O}', d) = \max_{C' \text{ max. chain in } \mathcal{O}'} \sum_{a \in C'} d_a \quad (61)$$

$$= \max_{C \text{ max. chain in } \mathcal{O}} \sum_{a \in C} d_a^v \quad (62)$$

$$= t(\mathcal{O}, d^v). \square \quad (63)$$

Lemma 22. *Let \mathcal{O} be a successful attack on T . Then there exists a successful attack \mathcal{O}' on T^v with $t(\mathcal{O}, d) \geq t(\mathcal{O}', d^v)$.*

Proof. Let $\mathcal{O} = (A, \prec)$. If \mathcal{O} does not reach v , define $\mathcal{O}' = (A', \prec')$ by

$$A' = A \setminus B_v, \quad (64)$$

$$\prec' = \prec|_{A'}. \quad (65)$$

Then \mathcal{O}' is a successful attack on both T^v and T . Since $\mathcal{O}' \leq \mathcal{O}$ one has $t(\mathcal{O}', d^v) = t(\mathcal{O}', d) \leq t(\mathcal{O}, d)$. Now suppose \mathcal{O} reaches v ; then $\mathcal{O}|_{B_v}$ is a successful attack on T_v . Define another attack $\mathcal{O}_1 = (A_1, \prec_1)$ on T by $A_1 = A \setminus A$ and $a \prec_1 a'$ if and only if one of the following holds:

- $a, a' \in A \setminus B_v$ and $a \prec a'$;
- $a, a' \in B_v$ and $a \prec a'$;
- $a \in A \setminus B_v$, $a' \in B_v$ and $a \prec b$ for all $b \in B_v \cap A$;
- $a \in B_v$, $a' \in A \setminus B_v$ and $b \prec a'$ for all $b \in B_v \cap A$.

\mathcal{O}_1 is the attack obtained from \mathcal{O} by removing all relations between elements of $A \setminus B_v$ and elements of B_v that are not shared with all elements of B_v . Since v is a module, the constraints on \prec regarding relations between elements of B_v and elements of $A \setminus B_v$ in Definition 3 are the same for all elements of B_v . Hence we find that \mathcal{O}_1 is successful because \mathcal{O} is. Furthermore, $\mathcal{O}_1 \leq \mathcal{O}$, so $t(\mathcal{O}_1, d) \leq t(\mathcal{O}, d)$.

Define an attack $\mathcal{O}' = (A', \prec')$ on T^v by $A' = A_1 \setminus B_v \cup \{\tilde{v}\}$ and $a \prec' a'$ if and only if one of the following holds:

- $a, a' \in A_1 \setminus B_v$ and $a \prec_1 a'$;
- $a \in A_1 \setminus B_v$, $a' = \tilde{v}$, and $a \prec_1 b$ for all $b \in A_1 \cap B_v$;
- $a = \tilde{v}$, $a' \in A_1 \setminus B_v$, and $b \prec_1 a'$ for all $b \in A_1 \cap B_v$.

Since \mathcal{O}_1 is successful on T we find that \mathcal{O}' is successful on T^v . Furthermore, the set of maximal chains of \mathcal{O}_1 is obtained by taking a maximal chain of \mathcal{O}' and replacing \tilde{v} (if it occurs) with a maximal chain of $\mathcal{O}|_{B_v}$. It follows that

$$\begin{aligned} t(\mathcal{O}_1, d) &= \max(M_1, M_2 - d_v^v + t(\mathcal{O}|_{B_v}, d_v)) \end{aligned} \quad (66)$$

where

$$M_1 = \max_{C \text{ max. chain of } \mathcal{O}'} \sum_{a \in C} d_a, \quad (67)$$

$$M_2 = \max_{C \text{ max. chain of } \mathcal{O}'} \sum_{a \in C} d_a. \quad (68)$$

By definition of d_v^v one has $t(\mathcal{O}|_{B_v}, d_v) \geq d_v^v$. Since $t(\mathcal{O}', d^v) = \max\{M_1, M_2\}$, it follows that $t(\mathcal{O}_1, d) \geq t(\mathcal{O}', d^v)$. Since we already know that $t(\mathcal{O}, d) \geq t(\mathcal{O}_1, d)$, we get $t(\mathcal{O}, d) \geq t(\mathcal{O}', d^v)$. \square

Proof of Theorem 14. By Lemma 21 one has $\text{mt}(T, d) \leq \text{mt}(T^v, d^v)$, and by Lemma 22 one has $\text{mt}(T, d) \geq \text{mt}(T^v, d^v)$. \square