Synthesis of Optimal Defenses for System Architecture Design Model in MaxSMT

Baoluo Meng, Arjun Viswanathan, William Smith, Abha Moitra, Kit Siu, and Michael Durling

GE Research, Niskayuna NY 12309, USA
{baoluo.meng, arjun.viswanathan, william.d.smith, moitraa, siu, durling}@ge.com

Abstract. Attack-Defense Trees (ADTrees) are widely used in the security analysis of software systems. In this paper, we introduce a novel approach to analyze system architecture models via ADTrees and to synthesize an optimal cost defense solution using MaxSMT. We generate an ADTree from the Architecture Analysis and Design Language (AADL) model with its possible attacks, and implemented defenses. We analyze these ADTrees to see if they satisfy their cyber-requirements. We then translate the ADTree into a set of logical formulas, that encapsulate both the logical structure of the tree, and the constraints on the cost of implementing the corresponding defenses, such that a minimization query to the MaxSMT solver returns a set of defenses that mitigate all possible attacks with minimal cost. We provide an initial evaluation of our tool on a delivery drone system model which shows promising results.

Keywords: AADL system architecture model · Attack-defense tree analysis · Synthesis of optimal defenses · MaxSMT.

1 Introduction

System security has attracted worldwide attention as society has grown increasingly dependent on computer-based systems. To address the security concerns of systems, many risk analysis techniques have been introduced over the years in order to identify potential system failure and mitigate risks before the system is fielded. *Attack trees* [15] are a prominent methodology to visually depict the security vulnerabilities of a system. They have been used in the analysis of threats against systems in the fields of defense and aerospace. Attack trees capture attacks in a tree structure, where the root node represents the attacker's goal and child nodes refine the goal with details involved in achieving the goal. *Attack-defense trees (ADTrees)* [13] extend attack trees with the notion of defenses against attacks, with the objective of reducing the consequences of attacks. In an ADTree, defense and attack nodes are distinguished node types, and in addition to refinements of nodes via children of the same type of node, child nodes can be *counter-measures* of the opposite kind of parent node. Such trees are able to capture both the attacks and the defenses of a system in an adversarial model, and as such, can be used to analyze the sufficiency of attack mitigation techniques of the system.

Implementing defenses requires various amount of effort, time, and money. A challenging problem is to select a set of defenses that is able to mitigate all threats while incurring minimal cost of implementation. In this work, we use MaxSMT solvers to synthesize a set of optimal-cost defenses that mitigate all possible attacks on a system. A prototype was implemented in a tool called VERDICT — in conjunction with the model-based architectural analysis (MBAA) component, model-based architectural synthesis (MBAS) calculates a set of defenses for all known attacks at minimal cost. We make the following contributions in this paper.

– We describe an approach that converts an AADL system architecture model to an attack-defense tree and an evaluation of these ADTrees in terms of a set of cyber-requirements.
– We encode the ADTree along with the costs of implementing defenses as a MaxSMT problem so that the solver can find a least-cost defense solution that satisfies all requirements.
– We present the analysis and synthesis features in the VERDICT toolchain which provides an implementation of the above functionalities.
– We show an evaluation of the synthesis capability on a high-fidelity AADL model of a delivery drone system.

Section 2 presents our specifications of attacks, defenses, and attack-defense trees. Section 3 presents a translation of AADL models to ADTrees and our method of analyzing whether an ADTree satisfies its requirements. Section 4 describes the interaction with the SMT solver in determining a minimum-cost set of defenses for the system. Section 5 is an evaluation of the VERDICT toolchain on an AADL model of a delivery drone system. Section 6 discusses some related work and Section 7 discusses directions our work can move in the future, along with a summary.

2 Preliminaries

In this section, we formalize our problem and solution space. We consider attacks from the MITRE Common Attack Pattern Enumeration and Classification (CAPEC) library [1] that targets embedded systems and defenses (controls) from the National Institute of Standards and Technology's (NIST's) 800-53 security standard [2]. A toy drone example is used through the paper to illustrate various features.

2.1 Attack and Defense Specification

This work is based on two standards drafted by the Radio Technical Commission for Aeronautics (RTCA) — DO-326, the Airworthiness Security Process

Specification [3] and DO-356, the Airworthiness Security Methods and Considerations [4] – both providing guidance against threats to aircraft systems. The standards specify the acceptable level of risk corresponding to the level of severity of successful attacks. The severity of successful attacks is categorized into 5 levels based on their effects on the aircraft, crew, and passengers: Catastrophic, Hazardous, Major, Minor, and No Effect. These levels, along with the corresponding levels of risk acceptable in the system, are presented in the first two columns of Table 1.

Severity Level	**Acceptable Level of Risk**	**DAL**	**Score**	**Objective (W/Independence)**
Catastrophic	1×10^{-9}	A	9	66 (25)
Hazardous	1×10^{-7}	B	7	65 (14)
Major	1×10^{-5}	C	5	57 (2)
Minor	1×10^{-3}	D	3	28 (2)
No Effect	1	E	0	0 (0)

Table 1: Mapping between the severity of consequence, acceptable level of risk, design assurance level (DAL), DAL score (Score) and development objectives (with independence).

We represent by L the set of severity levels, and by ρ the set of acceptable risk levels. The top-level event of an ADTree represents the attacker's goal, which is measured in terms of confidentiality (`C`), integrity (`I`) and availability (`A`) of the outports of the system. The attacker's goal is to sabotage the system by compromising its components. Attacks on components are ultimately propagated to the outports of the system through internal connections. The system fails if the CIAs of its outports are compromised. To mitigate attacks, the system designer has to implement defenses in components with various degrees of rigor, which can prevent failure of the system. The previously mentioned standards map the rigor of defense implementation, called *Design Assurance Levels (DALs)*, to a security consideration score or DAL score — columns 3 and 4 of Table 1. DAL A is the highest rigor defense and E is the lowest. DALs originated in DO-178 and were reused in DO-254. These standards were developed to ensure that software and complex hardware were developed with enough rigor and could be proved to be absent of bugs with potentially severe consequences. To bring the system within an acceptable risk level of attacks, and to prevent the system from failing with the associated severity level, its developers need to implement the component to the respective DAL score and meet appropriate development objectives from the fifth column. For example, if the failure of software can have a Catastrophic consequence, one must show compliance to 66 objectives as part of the software development process, 25 of which need to be performed by independent developers. The implementation of defenses incurs efforts and cost,

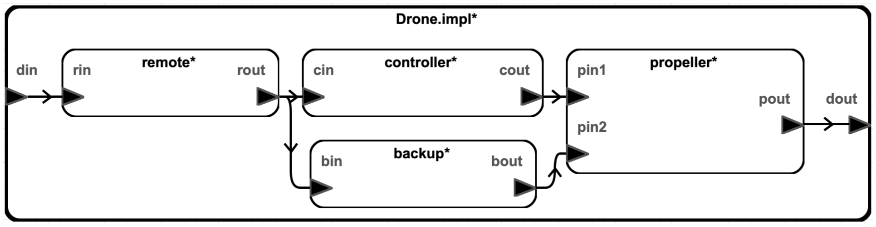


Fig. 1: The AADL structure diagram for the toy drone model

which increase with the DAL score, and the goal of this work is to synthesize a set of defenses that mitigate all attacks at an optimal (minimal) cost.

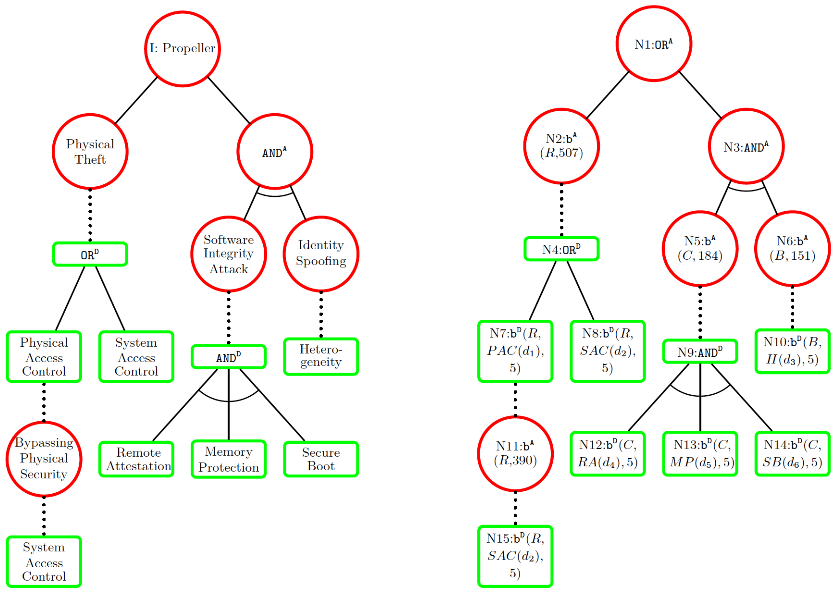
We will use CIA to denote the set consisting of the properties confidentiality ($\mathtt{C}$), integrity ($\mathtt{I}$) and availability ($\mathtt{A}$) ($CIA = \{\mathtt{C}, \mathtt{I}, \mathtt{A}\}$), and DAL, the set of possible DAL scores ($DAL = \{0, 3, 5, 7, 9\}$). We also consider the sets: A of possible attacks, D of possible defenses, and S of components of a system.

2.2 Attack-Defense Trees

ADTrees are rooted, labeled, finite trees that represent scenarios of security attacks against a system, and the countermeasures taken against these attacks. The nodes of an ADTree are either *attack nodes* — represented as red circles — or *defense nodes* — represented as green rectangles, and the nodes are labeled either with attack or defense goals, or with logical gates that connect these goals. A node's children represent either *refinements* (represented by straight line edges) of the same node type or *countermeasures* (dotted line edges) of the opposite node type. Refinements can either be conjunctive, (denoted in diagrams with an arc below the parent node) in which case, all the refinements' goals must be achieved for the parent's goal to be achieved; or disjunctive, where at least one of the refined goals must be achieved for the parent's goal to be achieved. Non-refined nodes (leaves) are called *basic actions*. The root of an ADTree represents the attacker's goal, which can be refined down to a logical formula over the basic actions (leaves) by expanding on the refinements performed by nodes (conjunctions and disjunctions).

Example 1. Consider the following model that abstracts a simple drone. The model is represented using an architecture diagram in Figure 1. A remote control allows the user to direct the drone. The drone consists of a controller that implements its logic, and a propeller that helps the drone move. As a security measure, the drone consists of a backup controller which implements a much simpler logic than the main controller. The user of the remote may invoke the single functionality of the backup which brings the drone back to its base.

A wireless connection connects the remote to the controller and to the backup, both of which have a wired connection to the propeller. Figure 2a shows the ADTree that models the attacks and defenses of the drone system, supposing



(a) ADTree of drone system (b) Labeled ADTree of drone system

Fig. 2: Example toy drone system

that we care about the integrity of the drone's propeller, that is, we want the propeller to move as instructed by the remote, and return back safely to the owner if that isn't feasible. Consider the following attacks:

1. Physical Theft Attack (CAPEC–507) on the remote.
2. A combination of a Software Integrity Attack (CAPEC–184) on the controller, and an Identity Spoofing Attack (CAPEC–151) on the backup controller.

Either Physical Access Control or System Access Control of the remote can defend against Physical Theft, but CAPEC–390 (Bypassing Physical Security) is a *dependent attack* that becomes applicable once Physical Access Control is implemented, and can only be defended against by implementing System Access Control. Three defenses — Remote Attestation, Memory Protection, and Secure Boot — are necessary for the controller to mitigate CAPEC-184 and Heterogeneity alone implemented on the backup controller can protect it against the identity spoofing attack. In Figure 2b, we label the nodes, attacks, and defenses, and also use the notation from our ADTree definition (Definition 1). We also give label R for the remote sub-system, C for the controller, and B for the backup controller. All defenses are implemented to DAL-score 5.

Definition 1. *An* ADTree T *is generated by the following grammar.*

$$\begin{aligned}
T &\to T^{\mathtt{A}} \mid T^{\mathtt{D}} \\
T^{\mathtt{A}} &\to \mathtt{b}^{\mathtt{A}}(s,a) \mid \mathtt{OR}^{\mathtt{A}}(T^{\mathtt{A}},\ldots T^{\mathtt{A}}) \mid \mathtt{AND}^{\mathtt{A}}(T^{\mathtt{A}},\ldots T^{\mathtt{A}}) \mid \mathtt{C}^{\mathtt{A}}(T^{\mathtt{A}},T^{\mathtt{D}}) \\
T^{\mathtt{D}} &\to \mathtt{b}^{\mathtt{D}}(s,d,\delta) \mid \mathtt{OR}^{\mathtt{D}}(T^{\mathtt{D}},\ldots T^{\mathtt{D}}) \mid \mathtt{AND}^{\mathtt{D}}(T^{\mathtt{D}},\ldots T^{\mathtt{D}}) \mid \mathtt{C}^{\mathtt{D}}(T^{\mathtt{D}},T^{\mathtt{A}})
\end{aligned}$$

Superscripts `A` *and* `D` *represent attack and defense entities, respectively.* T *represents terms or trees,* `OR` *encapsulates disjunctive refinements of a node,* `AND` *represents conjunctive refinements of a node, and* `C` *encapsulates the action of a node and its countermeasure.* `b` *represents basic actions — for attack nodes, they are parameterized by a component and an attack, and for defense nodes, they are parameterized by a component, a defense, and implemented DAL-score. An attack tree, denoted* $T^{\mathtt{A}}$*, is an ADTree with root of type* `A` *and a defense tree, denoted* $T^{\mathtt{D}}$*, is a tree with root of type* `D`*. We define a function* `root` *that returns the root node of an ADTree.*

We use an inductive definition of ADTree in Definition 1 from Kordy et al. [14]. Defenses are implemented to a particular DAL-score, and the defense nodes (that are basic actions) are parameterized by this DAL-score, along with the component that they defend and the defense itself. DAL-scores can only take values from column 4 of Table 1. Although our definition allows for any kind of ADTree, in practice, we only consider ADTrees with attack root nodes. This suits our goal of using ADTrees to analyze the attacker's actions. An interesting feature of our ADTrees is that we allow repetition of defense and attack nodes. That is, multiple $\mathtt{b}^{\mathtt{A}}$ and $\mathtt{b}^{\mathtt{D}}$ nodes in our trees can have the same label ((s,a) or (s,d,δ)). The only restriction we place is that when two $\mathtt{b}^{\mathtt{A}}$ or $\mathtt{b}^{\mathtt{D}}$ nodes have the same label, their child-node structure must be identical.

3 ADTree Analysis

In this section, we describe how our tool uses ADTrees to analyze a system architecture modeled using AADL (Architecture Analysis and Design Language) [10], which provides a framework and language for early analyses of a system's architecture with respect to performance-critical properties. Our tool builds an ADTree from an AADL model, a specification of possible attacks, possible defenses, implemented defenses, and cyber-requirements to satisfy. This tree is evaluated in terms of the likelihood of success of an arbitrary attacker, given a set of defenses.

3.1 Defense Models

Within VERDICT, the analysis of the AADL model receives information primarily from the Security Threat Evaluation and Mitigation (STEM) component [17]. STEM identifies possible CAPEC attacks, possible NIST-800-53 defenses and defenses implemented in the components of the system. STEM provides this data in the form of a defense model $\mathbb{M}$ with two types of relations: an implemented defense model $\mathbb{M}_{\mathbb{I}}$ and an applicable defense model $\mathbb{M}_{\mathbb{A}}$.

A *defense model* $\mathbb{M}$ is a relation containing tuples that relate components of a system to defense–DAL-score pairs, and attack–CIA pairs. Each tuple signifies the applicability of an attack (if any) to a component, and either the applicability or implementation of a set of defenses to the same component to respective DAL-scores. We distinguish 3 types of defense models, and define two of them as follows. The third, the synthesized defense model, is defined later.

1. An *implemented defense model* $\mathbb{M}_{\mathbb{I}}$ represents defenses currently implemented in the system. We say $(s, a, \gamma, \Delta) \in \mathbb{M}_{\mathbb{I}}$ iff in component s, γ attack a is applicable, and for each $(d, \delta) \in \Delta$, defense d is implemented to DAL-score δ, where $s \in S, a \in A, \gamma \in CIA, d \in D, \delta \in DAL$.
2. An *applicable defense model* $\mathbb{M}_{\mathbb{A}}$ represents defenses applicable in the system. We say $(s, a, \gamma, \Delta) \in \mathbb{M}_{\mathbb{A}}$ iff in component s, γ attack a is applicable, and for each $(d, \delta) \in \Delta$, defense d is applicable to DAL-score δ.

In the defense models that it provides, STEM guarantees that basic action nodes with the same labels have the same sub-trees, as required by our mechanism in section 2.2.

Example 2. Consider the ADTree from Example 1 in terms of the following cyber requirement: $q : \ (d_{out} : \mathtt{I})$ with Major (1×10^{-5}) severity level. In other words, q requires the integrity of outport d_{out} to be resilient to attacks of Major severity level. While the tree from Figure 2 models the applicable defense model, we show two different implementations of defenses. The applicable defense model $\mathbb{M}_{\mathbb{A}}$ (Figure 2) consists of the following set of tuples:
$\{(R, CAPEC\text{–}507, \mathtt{I}, \{(d_1, 5)\}), (R, CAPEC\text{–}507, \mathtt{I}, \{(d_2, 5)\}),$
$(B, CAPEC\text{–}151, \mathtt{I}, \{(d_3, 5)\}), (C, CAPEC\text{–}184, \mathtt{I}, \{(d_4, 5), (d_5, 5), (d_6, 5)\}\}.$
The implemented defense model $\mathbb{M}_{\mathbb{I}}$ consists of the following set of tuples.
$\{(R, CAPEC\text{–}507, \mathtt{I}, \{(d_2, 7)\}), (C, \ CAPEC\text{–}184, \mathtt{I}, \{(d_4, 9), (d_5, 7), (d_6, 5)\})\}$
A second implementation is specified using $\mathbb{M}'_{\mathbb{I}}$ as follows.
$\{(R, CAPEC\text{–}507, \mathtt{I}, \{(d_1, 5)\}), \{(C, CAPEC\text{–}184, \mathtt{I}, \{(d_4, 3)\}),$

3.2 ADTree Construction

In this subsection, we briefly explain the ADTree construction algorithm `ADTree`. The full algorithm is provided in Appendix A. `ADTree` operates on the following parameters:

1. *mod*, the AADL model of a system, that specifies ports P, connections C, components S, and cyber-relations R between ports of the system, where cyber relations are internal to a component, and specify how the conjunctions or disjunctions CIA vulnerabilities of inports propagate to the CIA vulnerabilities of outports. An example cyber relation is `in1:I or in2:I => out1:I`, which states that compromise of the integrity of `in1` or `in2` would lead to the compromise of the integrity of `out1`.
2. Q, the set of cyber requirements, where each cyber-requirement q is a logical formula over (p, γ) atoms, $p \in P, \gamma \in CIA$, with a corresponding level of severity $l \in L$.

3. $\mathbb{M}$, a defense model.

and returns an ADTree T corresponding to the requirements in Q on the AADL model *mod* considering attacks and defenses in $\mathbb{M}$. The details about the language (VERDICT annex) specifying cyber relations and cyber requirements can be found in Meng et al. [16].

The algorithm constructs an ADTree for each requirement, and combines them using an $\texttt{OR}^{\texttt{A}}$ node. The tree for each requirement is constructed by backtracing through the connections in the model, starting at the relevant outport, and checking for the CIA property specified by the requirement. During backtracing the logical structure of the requirements and cyber-relations are reflected in the logical structures of the corresponding ADTrees. Finally, the algorithm also runs through the constructed tree to remove any redundant nodes.

Example 3. Using the cyber requirements Q consisting of the single requirement $q : (\texttt{d}_{\texttt{out}} : \texttt{I})$ with Major severity level and the applicable defense model from Example 1, `ADTree` can construct the ADTree in Figure 2b.

3.3 ADTree Evaluation

An ADTree represents the goal of the attacker, and an evaluation of the tree specifies the likelihood of success of the attacker in achieving this goal. The evaluation of the ADTree was introduced by Siu et al. [18]. We formalize it as a recursive function M as follows, and call it *measure*.

$$\begin{aligned}
M(T) \coloneqq {} & \text{match } T \text{ with} \\
& \mid \texttt{b}^{\texttt{A}}(s,a) \rightarrow 1 \\
& \mid \texttt{b}^{\texttt{D}}(s,d,\delta) \rightarrow 1e^{-\delta} \\
& \mid \texttt{OR}^{\texttt{A}}(T_1,\ldots,T_n) \rightarrow \texttt{max}(M(T_1),\ldots,M(T_n)) \\
& \mid \texttt{OR}^{\texttt{D}}(T_1,\ldots,T_n) \rightarrow \texttt{min}(M(T_1),\ldots,M(T_n)) \\
& \mid \texttt{AND}^{\texttt{A}}(T_1,\ldots,T_n) \rightarrow \texttt{min}(M(T_1),\ldots,M(T_n)) \\
& \mid \texttt{AND}^{\texttt{D}}(T_1,\ldots,T_n) \rightarrow \texttt{max}(M(T_1),\ldots,M(T_n)) \\
& \mid \texttt{C}^{\texttt{A}}(\texttt{b}^{\texttt{A}}(s,a),T^{\texttt{D}}) \rightarrow \texttt{min}(M(\texttt{b}^{\texttt{A}}(s,a)),M(T^{\texttt{D}})) \\
& \mid \texttt{C}^{\texttt{D}}(\texttt{b}^{\texttt{D}}(s,d,\delta),T^{\texttt{A}}) \rightarrow \texttt{max}(M(\texttt{b}^{\texttt{D}}(s,d,\delta)),M(T^{\texttt{A}}))
\end{aligned}$$

Basic attack nodes are always assigned a value of 1 for likelihood of a successful attack. Assigning a number to the level of attack is quite difficult and would hold true for only a short period of time, and not for the lifetime of a system. According to Javaid et al. [12], the issue with deciding the likelihood of various attacks is that "the risk values may be different for different researchers according to the information available and level of analysis. Hence, more emphasis should be put on countermeasures for threats which receive high priority." Thus, we chose to assume a worst-case likelihood for attacks (from the point of view of defending the system) and give more fine-grained scores for defenses.

Satisfaction A cyber-requirement q specifies the severity level l of a CIA of an outport of the system. A defense model $\mathbb{M}$ corresponding to AADL model mod *satisfies* q, $\mathbb{M} \vdash q$, if $M(T_q) \leq \rho$ where $T_q = \texttt{ADTree}(mod, q, \mathbb{M})$ and ρ is the acceptable level of risk corresponding to l from Table 1. In this case, we also say that T_q satisfies q, or $T_q \vdash q$. Intuitively, implementing the defenses from $\mathbb{M}$ in mod results in an ADTree whose attacks are mitigated. Satisfaction of a requirement by a model (resp. ADTree) is naturally extended to a set of requirements.

$$\mathbb{M} \vdash Q \quad \text{ if } \forall q \in Q,\ \mathbb{M} \vdash q$$

A tree constructed from $\mathbb{M}_{\mathbb{A}}$ satisfies its requirements, by definition, while one constructed from $\mathbb{M}_{\mathbb{I}}$ may or may not.

Example 4. The following are the evaluations of the ADTrees from our applicable and implemented defense models. $M(\texttt{ADTree}(mod, q, \mathbb{M}_{\mathbb{A}})) = 1 \times 10^{-5}$; $M(\texttt{ADTree}(mod, q, \mathbb{M}_{\mathbb{I}})) = 1 \times 10^{-7}$; $M(\texttt{ADTree}(mod, q, \mathbb{M}'_{\mathbb{I}})) = 1$.

Thus, $\mathbb{M}_{\mathbb{A}}$ and $\mathbb{M}_{\mathbb{I}}$ satisfy q while $\mathbb{M}'_{\mathbb{I}}$ does not. An evaluation using the applicable defense model is always within the level of severity corresponding to the requirement. The evaluation of $\mathbb{M}'_{\mathbb{I}}$ shows that the implementation does not succeed in stopping the attacker because the bypassing physical security attack is not defended at all, and neither of CAPEC–184 and CAPEC–151 are defended sufficiently. $\mathbb{M}_{\mathbb{I}}$, on the other hand, is able to satisfy the requirement.

4 ADTree Synthesis

While the goal of analysis is to construct an ADTree from an AADL model and determine whether the cyber-requirements are satisfied (alternatively, whether the attacks corresponding to the ADTree are mitigated), synthesis constructs an optimal set of defenses based on a cost model for these defenses and (possibly) on the currently implemented defenses.

We define the concepts of synthesized defense models, and cost models.

Definition 2. ***Synthesized Defense Model*** *A synthesized defense model* $\mathbb{M}_{\mathbb{S}}$ *is the set of optimal defenses to implement, output by synthesis. If* $(s, a, \gamma, \Delta) \in \mathbb{M}_{\mathbb{S}}$*, then for each* $(d, \delta) \in \Delta$*, the implementation of defense* d *to DAL-score* δ *in* s *is part of the optimal solution to mitigate* γ *attack* a*.*

Definition 3. ***Cost Model*** *The cost model* $\mathbb{C}$ *associates a cost with each component–defense–DAL-score triple from the tuples in a defense model. Given defense* d*, sub-component* s*, and DAL-score* δ*, the cost of implementing* d *in* s *to* δ *is the non-negative real number represented by* $\mathbb{C}(s, d, \delta)$*.*

$$\mathbb{C} : S \times D \times DAL \to \mathbb{R}_{\geq 0}$$

We define the cost model of a defense model $\mathbb{M}$ *as follows.*

$$\mathbb{C}(\mathbb{M}) = \forall (s, a, \gamma, \Delta) \in \mathbb{M}, \sum_{(d,\delta) \in \Delta} \mathbb{C}(s, d, \delta)$$

The only restriction we place on cost models is that costs must be monotonically increasing with respect to the DAL-scores, that is, $\delta_i > \delta_j \rightarrow \mathbb{C}(s, d, \delta_i) \geq \mathbb{C}(s, d, \delta_j)$, for any $s \in S$, $d \in D$, and $\delta_i, \delta_j \in DAL$. This reflects the expectation that higher DALs are more expensive to implement (or at least, not cheaper). The synthesis problem seeks an optimal solution with respect to $\mathbb{C}$. The cost may represent financial cost, time required for implementation, or perhaps some compound or abstract definition of cost. For simplicity, one might consider a cost model that assigns the DAL-score as the cost of a component–defense–DAL-score triple, $\mathbb{C}(s, d, \delta) = \delta$ (for arbitrary s, d, and δ).

Example 5. Recollect the requirement q for our example drone system:

$$q: \ (d_{out} : \mathtt{I}) \text{ with Major } (1 \times 10^{-5}) \text{ severity level}$$

As we informally stated in Example 4, one implementation of the defenses doesn't satisfy q, another does, and the applicable defenses also satisfy q, by definition.

$$\mathbb{M}_{\mathbb{A}} \vdash q$$
$$\mathbb{M}_{\mathbb{I}} \vdash q$$
$$\mathbb{M}'_{\mathbb{I}} \nvdash q$$

Now, we define a cost model $\mathbb{C}$ for the drone system.

$$\mathbb{C}(s, d, \delta) = \begin{cases} 2\delta, & \text{for } (Remote, d_1, \delta) \\ 2\delta, & \text{for } (Remote, d_2, \delta) \\ 4\delta, & \text{for } (Backup, d_3, \delta) \\ 2\delta, & \text{for } (Controller, d_4, \delta) \\ 2\delta, & \text{for } (Controller, d_5, \delta) \\ 3\delta, & \text{for } (Controller, d_6, \delta) \\ \delta, & \text{otherwise} \end{cases}$$

□

4.1 Synthesis Problem Statement

The goal of synthesis is to construct a set of defenses to mitigate all attacks with the least cost. We distinguish 3 cases to synthesize solutions for.

1. Ignore implemented defenses. In this case, the job of synthesis is to synthesize a defense model $\mathbb{M}_{\mathbb{S}}$ from scratch such that $\mathbb{M}_{\mathbb{S}} \vdash Q$ and $\mathbb{C}(\mathbb{M}_{\mathbb{S}})$ is minimal. This case finds a globally minimal solution, in the sense that every other solution which mitigates the attack-defense tree must have a cost greater than or equal to the cost of $\mathbb{C}(\mathbb{M}_{\mathbb{S}})$. It resembles the early design phase of a system, when defenses have not yet been implemented.
2. Use implemented defenses. There are two possible cases to consider here.
   (a) $\mathbb{M}_{\mathbb{I}} \vdash Q$. In other words, all possible attacks are mitigated and the requirements in Q are satisfied by the implemented defenses in $\mathbb{M}_{\mathbb{I}}$. In this case, synthesis tries to optimize the implemented defenses. $\mathbb{M}_{\mathbb{S}}$ is an optimization of $\mathbb{M}_{\mathbb{I}}$ using any combination of:

i. eliminating unnecessary defenses — removing tuples from $\mathbb{M}_{\mathbb{I}}$
ii. downgrading current defenses — replacing $(s, a, \gamma, \{(d, \delta_i), \Delta_R\})$ in $\mathbb{M}_{\mathbb{I}}$ with $(s, a, \gamma, \{(d, \delta_j), \Delta_R\})$ such that $\delta_j < \delta_i$

This case resembles a situation where successful defenses have already been implemented, but can be downgraded or removed to save costs. Here, we restrict addition of new defenses to save costs.

(b) $\mathbb{M}_{\mathbb{I}} \not\models Q$. In other words, the requirements in Q are not satisfied by the implemented defenses in $\mathbb{M}_{\mathbb{I}}$. In this case, synthesis corrects the implemented defenses with the least amount of change possible. $\mathbb{M}_{\mathbb{S}}$ is a modification of $\mathbb{M}_{\mathbb{I}}$ using some combination of:

i. implementing new defenses — adding triples to $\mathbb{M}_{\mathbb{I}}$
ii. upgrading current defenses — replacing $(s, a, \gamma, \{(d, \delta_i), \Delta_R\})$ in $\mathbb{M}_{\mathbb{I}}$ with $(s, a, \gamma, \{(d, \delta_j), \Delta_R\})$ such that $\delta_j > \delta_i$

A real-life application of this situation is one where defenses have been implemented, unsuccessfully, and need to be improved to mitigate attacks, at minimal additional cost. The already implemented defenses are considered sunk costs that cannot be recovered and thus are not downgraded or removed.

4.2 MaxSMT Encoding for Synthesis

The problem of optimizing the defense costs is stated as a MaxSMT problem and sent to Z3's MaxSMT solver [8]. The input to the MaxSMT solver is an SMT-LIB [6] script (with some extensions for the optimization commands) that includes (i) declarations of variables, (ii) assertions of formulas, and, (iii) an expression over the variables to optimize, given the constraints asserted. Our MaxSMT encoding depends on the case we are encoding from Subsection 4.1.

In all 3 cases, we do the following. For each component–defense pair (s, d) $((s, \_, \_, \{(d, \_), \_\}) \in \mathbb{M}_{\mathbb{A}})$, we declare a variable $v_{s,d}$ which stands for the real number representing the synthesized cost of implementing defense d in component s to a particular DAL δ (for each $(s, \_, \_, \{(d, \delta), \_\})$ that we care about, we add a constraint on $v_{s,d}$, as we will show). Since we allow repeated labels, notice that during the creation of these $v_{s,d}$ variables, multiple nodes in the tree might necessitate the creation of the same variable. Some mechanism, such as a hash table, would have to check that variable declarations aren't repeated in the SMT script. Constraints, however, can be repeated.

For each variable $v_{s,d}$, we assert that the cost is non-negative. Then, we encode the $\texttt{ADTree}(\mathit{mod}, Q, \mathbb{M}_{\mathbb{A}})$ as an assertion, where mod is the AADL model of the system, Q is the set of requirements, and $\mathbb{M}_{\mathbb{A}}$ is the applicable defense model. Since $\mathbb{M}_{\mathbb{A}}$ satisfies Q, this assertion sets a baseline on the synthesized model. The following function F converts an ADTree to a quantifier-free first-order formula, which is asserted.

$$\begin{aligned}
F(T) \coloneqq\ & \text{match } T \text{ with} \\
& \mid \mathtt{OR}^{\mathtt{A}}(T_1, \ldots, T_n) \to F(T_1) \wedge \ldots \wedge F(T_n) \\
& \mid \mathtt{OR}^{\mathtt{D}}(T_1, \ldots, T_n) \to F(T_1) \vee \ldots \vee F(T_n) \\
& \mid \mathtt{AND}^{\mathtt{A}}(T_1, \ldots, T_n) \to F(T_1) \vee \ldots \vee F(T_n) \\
& \mid \mathtt{AND}^{\mathtt{D}}(T_1, \ldots, T_n) \to F(T_1) \wedge \ldots \wedge F(T_n) \\
& \mid \mathtt{C}^{\mathtt{A}}(\mathtt{b}^{\mathtt{A}}(s, a), T^{\mathtt{D}}) \to F(\mathtt{b}^{\mathtt{A}}(s, a)) \vee F(T^{\mathtt{D}}) \\
& \mid \mathtt{C}^{\mathtt{D}}(\mathtt{b}^{\mathtt{D}}(s, d, \delta), T^{\mathtt{A}}) \to F(\mathtt{b}^{\mathtt{D}}(s, d, \delta)) \wedge F(T^{\mathtt{A}}) \\
& \mid \mathtt{b}^{\mathtt{A}}(s, a) \to \bot \\
& \mid \mathtt{b}^{\mathtt{D}}(s, d, \delta) \to v_{s,d} \geq \mathbb{C}(s, d, \delta)
\end{aligned}$$

$\mathtt{OR}^{\mathtt{A}}$ nodes are translated to conjunctions and $\mathtt{AND}^{\mathtt{A}}$ nodes to disjunctions because the ADTree is concerned with the success of the attacker while the MaxSMT encoding is concerned with the success of defending any possible attack. If an attacker needs a conjunction ($\mathtt{AND}^{\mathtt{A}}$) of attacks to succeed, it suffices from the defender's point of view to stop at least one of the attacks successfully, and hence the disjunction in the MaxSMT encoding. The reasoning for using conjunctions for $\mathtt{OR}^{\mathtt{A}}$ nodes is similar. Finally, we need to minimize the cost, which is done by using the `minimize` command in the SMT-LIB script over the sum of all variables representing the costs of defenses.

The variables declarations, assertions and the optimization command are common in all cases. Additions to the assertions are unique to each case of the problem statement and we consider each of the 3 cases (all assertions must be added before the optimization command in the script).

Case 1 Since we ignore implemented defenses, the constraint from $\mathbb{M}_{\mathbb{A}}$ – $(F(\mathtt{ADTree}(mod, Q, A, \mathbb{M}_{\mathbb{A}})))$ suffices to restrict the synthesized solution to one that mitigates all attacks. Additionally, the optimization command assures a global optimum.
Case 2(a) Since the implemented defenses satisfy the requirements, we assert constraints from $\mathbb{M}_{\mathbb{I}}$ — for each $(s, \_, \_, \{(d, \delta), \_\}) \in \mathbb{M}_{\mathbb{I}}$, assert $v_{s,d} \leq \mathbb{C}(s, d, \delta)$. We also restrict implementation of new defenses — for each $(s, a, \gamma\{(d, \delta), \Delta_R\})$ $\in \mathbb{M}_{\mathbb{A}}$ such that there exists no $(s, \_, \_, \{(d, \_), \_\}) \in \mathbb{M}_{\mathbb{I}}$, assert $v_{s,d} = 0$. Given the lower bounds from $\mathbb{M}_{\mathbb{A}}$, and the upper bounds from $\mathbb{M}_{\mathbb{I}}$, the MaxSMT solver finds the minimal cost solution, without adding any new defenses.
Case 2(b) Since the implemented defenses do not satisfy the requirements and the cost of implementing them is considered sunk, we assert them as lower bounds — for each $(s, \_, \_, \{(d, \delta), \_\}) \in \mathbb{M}_{\mathbb{I}}$, assert $v_{s,d} \geq \mathbb{C}(s, d, \delta)$. For defenses that don't work, the constraints from $\mathbb{M}_{\mathbb{A}}$ supersede the lower bound specified by the constraints from $\mathbb{M}_{\mathbb{I}}$.

The MaxSMT encoding for each case is summarized as follows.
Case 1:

$$\begin{aligned}
&\text{For each } s \in S, d \in D,\ \ \texttt{declare-var}\ v_{s,d}\\
&\text{For each } v_{s,d},\ \ \texttt{assert}\ v_{s,d} \geq 0\\
&\texttt{assert}\ F(\texttt{ADTree}(mod, Q, A, \mathbb{M}_{\mathbb{A}}))\\
&\texttt{minimize} \sum^{s\in S, d\in D} v_{s,d}
\end{aligned}$$

Case 2(a):

$$\begin{aligned}
&\text{For each } s \in S, d \in D,\ \ \texttt{declare-var}\ v_{s,d}\\
&\text{For each } v_{s,d},\ \ \texttt{assert}\ v_{s,d} \geq 0\\
&\texttt{assert}\ F(\texttt{ADTree}(mod, Q, A, \mathbb{M}_{\mathbb{A}}))\\
&\text{For each } (s, d, \delta) \in \mathbb{M}_{\mathbb{I}},\ \ \texttt{assert}\ v_{s,d} \leq \mathbb{C}(s, d, \delta)\\
&\text{For each } (s, d, \delta) \notin \mathbb{M}_{\mathbb{I}}, \texttt{assert}\ v_{s,d} = 0\\
&\texttt{minimize} \sum^{s\in S, d\in D} v_{s,d}
\end{aligned}$$

Case 2(b):

$$\begin{aligned}
&\text{For each } s \in S, d \in D,\ \ \texttt{declare-var}\ v_{s,d}\\
&\text{For each } v_{s,d},\ \ \texttt{assert}\ v_{s,d} \geq 0\\
&\texttt{assert}\ F(\texttt{ADTree}(mod, Q, A, \mathbb{M}_{\mathbb{A}}))\\
&\text{For each } (s, d, \delta) \in \mathbb{M}_{\mathbb{I}},\ \ \texttt{assert}\ v_{s,d} \geq \mathbb{C}(s, d, \delta)\\
&\texttt{minimize} \sum^{s\in S, d\in D} v_{s,d}
\end{aligned}$$

4.3 SMT Model Evaluation

All our calls to the MaxSMT solver are expected to be satisfiable. A solution where all possible defenses are implemented to the highest possible DAL would trivially satisfy the problem (while likely being unnecessarily expensive):

$$\forall (s, a, \gamma, d, \delta) \in \mathbb{M}_{\mathbb{A}},\ (s, a, \gamma, d, 9) \in \mathbb{M}_{\mathbb{S}}$$

The response from the solver varies in its optimization of defense cost. The variables $v_{s,d}$ in our SMT encoding model the cost of implementing defense d in component s to some DAL-score. Thus, when the SMT solver returns an optimal solution as a model, it returns an optimal cost for each component-defense pair. We need to build $\mathbb{M}_{\mathbb{S}}$ from this for which we need the DAL-score for each component-defense pair. We define the inverse cost function $\mathbb{C}^{-1}$ that given a component-defense-cost triple, returns the DAL-score to implement the defense to in the component. Since a component–defense pair could have the same cost for multiple DAL-scores (the monotonicity requirement does not prevent this), the inverse isn't over an injective function. We break ties by preferring higher DAL-scores, given equal costs.

$$\mathbb{C}^{-1}(c, s, d) = \mathtt{max}\{\delta_i \mid \mathbb{C}(s, d, \delta_i) = c\}$$

Thus, as a minimal cost solution, for each component s and defense d, the SMT solver returns a cost as the real value of $v_{s,d}$. $\mathbb{M}_{\mathbb{S}}$ is then constructed as follows.

$$\forall v_{s,d}, \forall a \in A, \text{ such that } (s, a, \gamma, \{(d, \delta), \Delta_R\}) \in \mathbb{M}_{\mathbb{A}}, \\ (s, a, \gamma, \{(d, \mathbb{C}^{-1}(v_{s,d}, s, d))\}) \in \mathbb{M}_{\mathbb{S}}$$

This is the minimal cost defense model synthesized by the MaxSMT solver.

Example 6. We construct synthesized defense models from the satisfiable solution returned by the SMT solver for the toy drone system using the cost model in Example 5 as follows.

- **Case 1** Without any additional restrictions, the SMT solver returns values 0, 10, 20, 0, 0 and 0 respectively, for $rd1$, $rd2$, $bd3$, $cd4$, $cd5$ and $cd6$, which are its recommended costs for the defenses. Applying the cost inverse function, we have the following DALs to implement the components to. $\mathbb{C}^{-1}(0, R, d_1) = 0$; $\mathbb{C}^{-1}(10, R, d_2) = 5$; $\mathbb{C}^{-1}(20, B, d_3) = 5$; $\mathbb{C}^{-1}(0, C, d_4) = 0$; $\mathbb{C}^{-1}(0, C, d_5) = 0$; $\mathbb{C}^{-1}(0, C, d_6) = 0$. This is an optimal cost solution unrestricted by any implementation constraints. The total cost is 30.
- **Case 2(a)** The SMT solver returns cost 0 for $rd1$ and $bd3$, cost 10 for $rd2$, $cd4$, and $cd5$, and 15 for $cd6$. Applying the cost inverse function, we have that d_1 and d_2 are to be implemented to DAL 0 and 5 in the remote, d_3 to DAL 0 in the backup controller, and d_4, d_5, and d_6 all to DAL 5 in the main controller. Here, since the implemented defenses already satisfy the requirement, new ones aren't added, and instead, synthesis suggests reductions. The global optimal solution would choose d_3 over d_4, d_5 and d_6, but since the latter are already implemented, synthesis only suggests DAL reductions where applicable (to d_4 and d_5). The total cost of the synthesized solution is 45, which is cheaper than the implementation which costs 61.
- **Case 2(b)** The SMT solver returns costs 10, 10, 20, 6, 0 and 0 for $rd1$, $rd2$, $bd3$, $cd4$, $cd5$, and $cd6$ which translate to DALs 5, 5, 5, 3, 0 and 0, respectively. Since the unsatisfactory defenses have already been implemented, their cost is considered a sunk cost (16 here). The SMT solver specifies what defenses need to be added to satisfy the requirements — d_2 and d_3 in this case. The total cost of the synthesized solution is 46.

Notice that the same defense can be applicable to a component to defend 2 different attacks. For example, system access control defends against both CAPEC–507 and CAPEC–390. Because our encoding doesn't take into account attacks (and it doesn't need to), once synthesis suggests to implement such a defense, we add all possible occurrences of it to $\mathbb{M}_{\mathbb{S}}$. While this redundance is necessary for soundness of the formalism, it can be ignored during implementation. In fact, it isn't necessary to map synthesized defenses to attacks they mitigate at all, we do it in the formalism just to be able to make synthesized solutions comparable with applicable and implemented solutions. □

5 Evaluation

A prototype of ADTree-based security analysis and synthesis was implemented in the VERDICT toolchain [16] [19], which is a plugin for the OSATE tool [5].

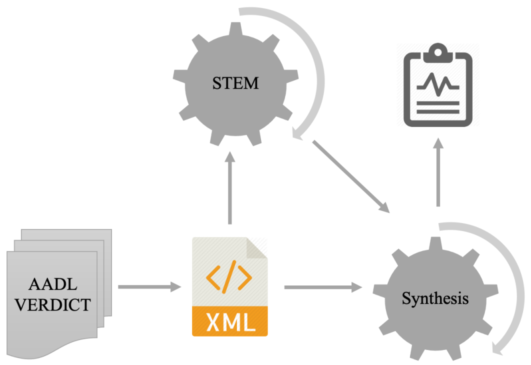


Fig. 3: The architecture of synthesis module in VERDICT.

The architecture of synthesis module in VERDICT is shown in Figure 3. The input to synthesis is a system architecture model in AADL annotated with cyber-relations and cyber requirements in VERDICT annex. The model will be translated into an intermediate representation in XML that will be further consumed by several modules in VERDICT. For synthesis, it will first be converted to the input to STEM tool, which will identify applicable attacks and also applicable or implemented defenses depending on the running mode of synthesis. The output from STEM and the XML model will then be leveraged by the synthesis module for attack-defense tree analysis and synthesis as described in Section 3 and Section 4.

We perform an evaluation on a high-fidelity architecture model of a delivery drone to demonstrate the capabilities of the tool. In addition, the tool was leveraged by Raytheon Technologies to evaluate on DoD applications development showing promising results [7]. The VERDICT tool is publicly available[1] .

A notional architecture for the delivery drone is shown in Figure 4. The AADL model[2] consists of 12 inter-connected components and is annotated with meta-level properties, defense properties, cyber relations and cyber requirements. Meta-level properties such as component type and pedigree, come built-in with the AADL model. Given this system, the STEM component of the VERDICT toolchain identifies possible CAPEC attacks and NIST 800-53 defenses. These attacks and defenses are fed to the synthesis tool for further processing. The

[1] VERDICT Tool GitHub: `https://github.com/ge-high-assurance/VERDICT`

[2] The Delivery Drone AADL Model: `https://github.com/baoluomeng/2022_NFM/tree/main/DeliveryDrone`

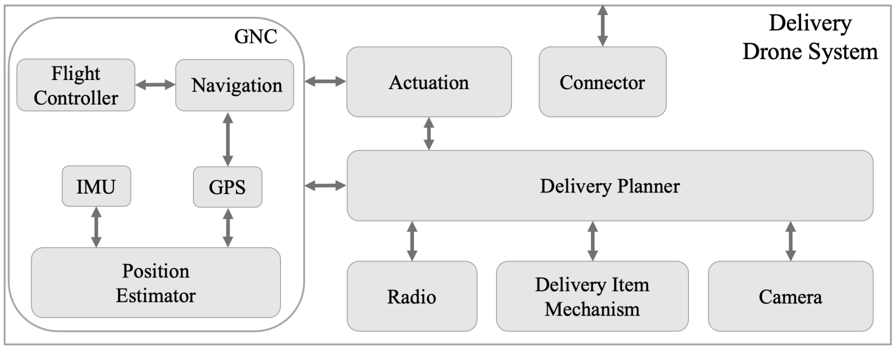


Fig. 4: A notional architecture diagram for the delivery drone model.

defense property is a numerical DAL-score from δ, and represents the rigor of implemented defense in each component of the system, and is used to construct $\mathbb{M}_{\mathbb{I}}$. Cyber relations and requirements are declared in an annex language to AADL – VERDICT. For example, one cyber requirement defines a successful mission of delivering a package to its destination, while requiring the drone to be resilient to malicious commands that attempt to obtain an improper delivery of a package. The requirement depends on the integrity of the output *delivery_status*, which is used as a starting point from which the system architecture is traced, to build the ADTree for analysis. Furthermore, the consequence of successful attack is Hazardous, requiring corresponding defenses to be implemented to at least DAL-score 7.

To demonstrate its optimizing capabilities, we invoke the Synthesis tool on the model for 3 cases (corresponding to the ones specified in 4.1)using the default cost model - one where the cost for each defense-DAL pair is the DAL score itself.

- **Case 1** The implemented defenses are ignored, and a *global* optimal solution is returned. Synthesis suggest a list of defenses with minimal costs to be implemented to DAL 7 so that all cyber requirements can be satisfied. The total cost for the implementation is 273.
- **Case 2** Implemented defenses are taken into consideration by Synthesis, and these don't satisfy the requirements. Synthesis suggests implementing two defenses for the *deliveryItemMechanism* component: Supply Chain Security and Tamper Protection, both to DAL 7, which would allow for the requirements to be satisfied. These would mitigate CAPEC-439 (Manipulation During Distribution) with an additional cost 14.
- **Case 3** Once the suggested defenses in case 2 are implemented in the model, they would be considered by Synthesis sufficient to satisfy all cyber requirements. In this case, Synthesis does "merit assignment" which is to suggest downgrades/removals of defenses (Case 2(a) from our problem statement) to save costs. For the delivery drone model, Synthesis suggest to remove Physical Access Control from the GPS component to save 7 units of cost.

6 Related Work

In our ADTrees, we use nodes with repeated labels — that is, there can exist multiple nodes in our tree that have the same label. Bossuat et al. [9] extend ADTrees to AD-DAGs to deal with repeated labels. In our work, by guaranteeing that these nodes will have the same child-structure, we are able to maintain the ADTree formalism, and also maintain soundness by handling repetitions during our SMT-encoding.

Fila et al. [11] and Kordy et al. [14] find an optimized set of defenses to mitigate an ADTree using integer linear programming. We use an SMT-based optimization approach, and also build our trees from AADL models of the system. Additionally, we are able to incorporate implementations of defenses that may or may not satisfy the requirements specified by the ADTree and suggest solutions based on these variations (cases 2(a) and 2(b) from Section 4.1). We use the formalism of attack-defense trees introduced by Kordy et al. [13] to specify our ADTrees.

7 Conclusion and Future Work

We propose a security analysis technique for system architecture designs via attack-defense trees, and a novel technique to synthesize optimal cost defenses for the components of a model. We translate the AADL model of a system into an ADTree, and encode this ADTree along with the cost of implementing its defenses into a MaxSMT query, such that a satisfying model of the SMT query is a minimum-cost defense for the system, that mitigates all applicable attacks. We utilize advancements in the ADTree literature and SMT technology, in building our formalism of the process of converting an AADL model to an ADTree and then to an optimization query to a MaxSMT solver. We provide an implementation of our technique as the Synthesis functionality in the VERDICT tool chain. One potential extension to our formalism and our tool is to allow a single defense to defend attacks over multiple components and connections – *extensibility* of defenses.

Acknowledgement & Disclaimer

Distribution Statement “A” (Approved for Public Release, Distribution Unlimited). This research was developed with funding from the Defense Advanced Research Projects Agency (DARPA). The views, opinions and/or findings expressed are those of the author and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

References

1. MITRE Common Attack Pattern Enumeration and Classification (CAPEC). https://capec.mitre.org/, accessed: 2022-03-21

2. National Institute of Standards and Technology 800-53. https://csrc.nist.gov/publications/detail/sp/800-53/rev-5/final, accessed: 2022-03-21
3. Radio Technical Commission for Aeronautics(RTCA) DO326 – Airworthiness Security Process Specification. https://www.rtca.org/, accessed: 2022-03-21
4. Radio Technical Commission for Aeronautics(RTCA) DO356 – Airworthiness Security Methods and Considerations. https://www.rtca.org/, accessed: 2022-03-21
5. The OSATE Tool. https://osate.org/about-osate.html (2021)
6. Barrett, C., Fontaine, P., Tinelli, C.: The Satisfiability Modulo Theories Library (SMT-LIB). `www.SMT-LIB.org` (2016)
7. Barzeele, J., Siu, K., Robinson, M., Suantak, L., Merems, J., Durling, M., Moitra, A., Meng, B., Williams, P., Prince, D.: Experience in designing for cyber resiliency in embedded dod systems. In: INCOSE International Symposium. vol. 31, pp. 80–94. Wiley Online Library (2021)
8. Bjørner, N., Phan, A.D., Fleckenstein, L.: νz - an optimizing smt solver. In: Baier, C., Tinelli, C. (eds.) Tools and Algorithms for the Construction and Analysis of Systems. pp. 194–199. Springer Berlin Heidelberg, Berlin, Heidelberg (2015)
9. Bossuat, A., Kordy, B.: Evil Twins: Handling Repetitions in Attack–Defense Trees: A Survival Guide. In: Liu, P., Mauw, S., , Stolen, K. (eds.) Graphical Models for Security. vol. LNCS, pp. 17–37. Springer, Santa Barbara, United States (Aug 2017), `https://hal.inria.fr/hal-01728782`
10. Feiler, P.H., Lewis, B., Vestal, S., Colbert, E.: An overview of the sae architecture analysis & design language (aadl) standard: A basis for model-based architecture-driven embedded systems engineering. In: Dissaux, P., Filali-Amine, M., Michel, P., Vernadat, F. (eds.) Architecture Description Languages. pp. 3–15. Springer US, Boston, MA (2005)
11. Fila, B., Wideł, W.: Exploiting attack–defense trees to find an optimal set of countermeasures. In: 2020 IEEE 33rd Computer Security Foundations Symposium (CSF). pp. 395–410 (2020). https://doi.org/10.1109/CSF49147.2020.00035
12. Javaid, A.Y., Sun, W., Devabhaktuni, V.K., Alam, M.: Cyber security threat analysis and modeling of an unmanned aerial vehicle system. In: 2012 IEEE Conference on Technologies for Homeland Security (HST). pp. 585–590. IEEE (2012)
13. Kordy, B., Mauw, S., Radomirović, S., Schweitzer, P.: Foundations of attack–defense trees. In: Degano, P., Etalle, S., Guttman, J. (eds.) Formal Aspects of Security and Trust. pp. 80–95. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)
14. Kordy, B., Widel, W.: How well can i secure my system? In: IFM (2017)
15. Mauw, S., Oostdijk, M.: Foundations of attack trees. In: Won, D.H., Kim, S. (eds.) Information Security and Cryptology - ICISC 2005. pp. 186–198. Springer Berlin Heidelberg, Berlin, Heidelberg (2006)
16. Meng, B., Larraz, D., Siu, K., Moitra, A., Interrante, J., Smith, W., Paul, S., Prince, D., Herencia-Zapana, H., Arif, M.F., et al.: Verdict: A language and framework for engineering cyber resilient and safe system. Systems **9**(1), 18 (2021)
17. Moitra, A., Prince, D., Siu, K., Durling, M., Herencia-Zapana, H.: Threat identification and defense control selection for embedded systems. SAE International Journal of Transportation Cybersecurity and Privacy **3**(11-03-02-0005), 81–96 (2020)
18. Siu, K., Herencia-Zapana, H., Prince, D., Moitra, A.: A model-based framework for analyzing the security of system architectures. In: 2020 Annual Reliability and Maintainability Symposium (RAMS). pp. 1–6. IEEE (2020)
19. Siu, K., Moitra, A., Li, M., Durling, M., Herencia-Zapana, H., Interrante, J., Meng, B., Tinelli, C., Chowdhury, O., Larraz, D., et al.: Architectural and behavioral

analysis for cyber security. In: 2019 IEEE/AIAA 38th Digital Avionics Systems Conference (DASC). pp. 1–10. IEEE (2019)

A ADTree Construction Algorithm

```
Input: $p \in P$, $\gamma \in CIA$, $\mathbb{M}$, Tree Location loc
Output: ADTree $T$
Algorithm ADTree_port($p$, $\gamma$, $\mathbb{M}$, loc)
  loc $\mapsto$ OR$^{A}$ (node*)
  for all incoming constituents $const$ to $p$ do
    Add child ADTree_const($const$, $\gamma$, $\mathbb{M}$, root($new\_tree$)) to node*
  end for
```

```
Input: $const \in S \cup C \cup R$, $\gamma \in CIA$, $\mathbb{M}$, Tree Location loc
Output: ADTree $T$
Algorithm ADTree_const($const$, $\gamma$, $\mathbb{M}$, loc)
  if $const$ is a component or a connection with inport $p_{in}$ and outport $p_{out}$ then
    loc $\mapsto$ OR$^{A}$ (node*)
    for all $a \in A \mid (const, a, \_, \_) \in \mathbb{M}$ do
      for all $(const, a, \gamma, ) \in \mathbb{M}$ do
        $(const, a, \gamma, \_) \in D_a$
        D $\leftarrow$ CrushTree (DTree($D_a$))
        if D is empty then
          Add b$^{A}$($const, a$) as child of node*
        else
          Add C$^{A}$(b$^{A}$($const, a$), D) as child of node*
        end if
      end for
      Add an additional child to node* and set it as loc
      ADTree_port($p_{in}$, $\gamma$, $\mathbb{M}$, loc)
    end for
  else if $const$ is a relation $F \rightarrow p_{out} : \gamma$ then
    loc $\mapsto$ OR$^{A}$ (node*)
    for all atom $p_{in} : \gamma \in F$ do
      ADTree_port($p_{in}$, $\gamma$, $\mathbb{M}$, root($new\_tree$))
    end for
    Connect all the new ADTrees using attack nodes that correspond to the logical
    connectives in $F$
  end if
```

Fig. 5: Mutually recursive functions ADTree_port and ADTree_const are used to construct an ADTree

The mutually recursive functions ADTree_port and ADTree_const from Figure 5 assist in constructing an ADTree. The notation loc $\mapsto$ node indicates that the node node is created at the location loc in the tree. An ADtree for a single (p, γ) atom from requirement q is constructed through the call ADTree_port(p, γ, $\mathbb{M}$, root(new_tree)). The function recursively scans through the ports within the architecture of the system, modeled in *mod*, while constructing the ADTree.

```
Input: Set of tuples D
Output: Defense tree $T^{D}$
Algorithm DTree($D$)
  Create an $OR^{D}$ (node*)
  for all ($const$, $a$, $\gamma$, $\Delta$) $\in D$ do
    Create an $AND^{D}$ node (node#)
    for all $(d, \delta) \in \Delta$ do
      Add $b^{D}(const, d, \delta)$ as a child of node#
    end for
    Add node# as child of node*
  end for
  Return the tree under node*
```

```
Input: ADTree $T$
Output: ADtree $T'$
Algorithm CrushTree($T$)
  Scan $T$ top-down, and
    1. If an $OR^{A}/OR^{D}$ node has a single child, replace the node by its child
    2. If an $OR^{A}/OR^{D}$ has no child, remove the $OR^{A}/OR^{D}$ node
```

Fig. 6: `DTree` is used to construct a defense tree from defenses, and `CrushTree` crushes an ADTree by removing unnecessary nodes

It calls `ADTree_const` when it encounters a *constituent* which is either a component, a connection, or a cyber-relation. `ADTree_const` calls `DTree` from Figure 6 which constructs a defense tree from a subset of the defense model under consideration. Once an ADTree is constructed, `CrushTree`, also from Figure 6, removes redundant nodes from the tree.

B Translation Soundness

Here, we argue about the soundness of the SMT encoding from Section 4.2. An attack-defense tree models the applicable attacks on the components and connections of a system, and the applicable/implemented defenses that mitigate these attacks. We encode one or more ADTrees as a set of constraints that we send to a MaxSMT solver. The solver may return one of 3 possible results:

1. `sat` - the problem instance is satisfiable. There exists a minimum cost solution which we can construct from the satisfiable model from the solver.
2. `unsat` - the problem instance is unsatisfiable. There exists no solution, given the constraints presented to the solver. The constraints are contradictory. We argue that our encoding never produces such a result, subject to certain assumptions.
3. `unknown` - the solver isn't able to give a conclusive response. When the solver fails, our method fails as well.

The following theorems relates the notion of satisfiability defined in Section 4 to the satisfiability of a formula, and argue about the soundness of our encoding. As a matter of notation, notice that an ADTree satisfies one or more requirements, so an ADTree is said to be satisfying if it mitigates its attacks and unsatisfying otherwise. A formula is satisfiable if there exists a satisfying model for its variables, and unsatisfiable if there doesn't exist any.

Theorem 1. *An unsatisfying tree — one that doesn't mitigate the attacks specified by the requirements — is translated to an unsatisfiable set of formulas.*
Proof. An ADTree can be made unsatisfying by two possible sources.

1. Undefended attack nodes
2. Attack nodes defended via insufficient defense nodes (DAL-score not high enough)

Case 1 For ADTree T with undefended attack nodes, $F(T)$ is unsatisfiable, and the proof is by induction on F. The most important case is the base case of $\mathtt{b}^{\mathtt{A}}$ nodes. A lone $\mathtt{b}^{\mathtt{A}}$ node (one that isn't encapsulated in a $\mathtt{C}^{\mathtt{A}}$ or $\mathtt{C}^{\mathtt{D}}$ node) is an undefended attack node, which F translates to $\perp$, an unsatisfiable formula. The $\mathtt{b}^{\mathtt{D}}$ node is always satisfying (explained later in the proof) and the other node combinators simply combine translations of child nodes using conjunctions and disjunctions.

Case 2 ADTrees defended via insufficient defense nodes will be converted to satisfiable formulas using F. This is because F models a constraint on the cost of implementing the defense, not its ability to mitigate an attack. A $\mathtt{b}^{\mathtt{D}}$ node is translated to an inequality between a Real-valued variable and its cost - a Real number. Independently, it is satisfiable, and can only be made unsatisfiable when considered along with a contradictory inequality/equality. However, the only kind of inequalities F adds are $\geq$ inequalities between a variable on the LHS and a constant on the RHS. Any two inequalities $v \geq x$ and $v \geq y$ over a variable v and constants x and y are satisfiable, since the satisfying model will contain a value for v which is greater than or equal to the maximum of x and y, and this value will also be greater than or equal to the other constant.

The conversion of an unsatisfying ADTree to a satisfiable query is a source of unsoundness. To prevent this from happening, our encoding only calls F on ADTrees built from applicable defenses — and these are satisfiable by definition.

The non-negativity constraints don't introduce unsoundness — since each of the $v_{s,d}$ variables represent the cost of implementing defense d in component s, we expect these values to be non-negative.

The constraints added by Case 2(a) of our problem statement (4.1) take a currently, satisfying ADTree, and specify upper bounds on the costs of the defenses in the ADTree. If $v_{s,d}$ is currently some value x, $v_{s,d} \leq x$ introduces no unsoundness, since this constraint still allows for $v_{s,d} = x$. Additionally, constraints are added restricting currently unimplemented defenses from being implemented. Since the current implementation is already satisfying in this case, not allowing new defenses doesn't introduce unsatisfiability.

The constraints added by Case 2(b) also don't introduce any unsoundness, since constraints of the form $v_{s,d} \geq x$ don't add unsoundness (same argument as for independent $\mathtt{b}^{\mathtt{D}}$ nodes above).

Therefore, the SMT encoding translates an unsatisfying ADTrees to unsatisfiable formulas.

Theorem 2. *A satisfying ADTree is translated to a satisfiable set of formulas, and the satisfying SMT model translates to a cheapest (satisfying) defense implementation.*

Proof. This reduces to proving that our encoding only encodes the necessary and sufficient conditions of the ADTree as a logical constraint. We argue this for F incuctively.

- Independently, an attack node $\mathtt{b}^{\mathtt{A}}$ is unsatisfying. It is translated to $\perp$ which is an unsatisfiable formula.
- Independently, a defense node $\mathtt{b}^{\mathtt{D}}$ is satisfying. It is translated to to an inequality constraint $v_{s,d} \geq x$ for some constant x, and this constraint is satisfiable, because our translation only introduces constraints of the form $v_{s,d} \geq x$ which can't contradict each other.
- A defense node with a countermeasure attack tree is satisfying if the defense node is satying, and the counter-measure attack tree is also satisfying. $\mathtt{C}^{\mathtt{D}}$ nodes are therefore encoded as conjunctions of their respective child nodes.
- An attack node with a countermeasure defense tree is translated to a disjunction of the translations of the attack node and the defense tree. The attack node is translated to $\perp$, so the satisfiability of $F(\mathtt{C}^{\mathtt{D}})$ is reduced to the satisfiability of the translation of its defense tree child.
- Within a defense tree, an $\mathtt{AND}^{\mathtt{D}}$ indicates that all child defense trees have to be implemented to defend against the attack in question, and $\mathtt{OR}^{\mathtt{D}}$ indicates that at least one of the child defense trees have to be implemented. As a consequence, $\mathtt{AND}^{\mathtt{D}}$ nodes are translated to conjunctions, and $\mathtt{OR}^{\mathtt{D}}$ nodes to disjunctions.
- For $\mathtt{AND}^{\mathtt{A}}$ and $\mathtt{OR}^{\mathtt{A}}$ nodes, the translations are reversed. An $\mathtt{AND}^{\mathtt{A}}$ node indicates that the attacker needs all the child attack trees to succeed for the parent to succeed. From the point of view of the defender (which is how an ADTree is analyzed), it suffices to defend at least one of the child attacks. Thus, an $\mathtt{AND}^{\mathtt{A}}$ node is translated to a disjunction. For similar reasons, an $\mathtt{OR}^{\mathtt{A}}$ node is translated to a conjunction.

F is applied to an ADTree constructed from the applicable defense model. The applicable defense model consists of the minimal DAL necessary for a defense to mitigate its respective attack. Thus, the defense nodes in the ADTree consist of the minimal DAL necessary for a particular defense to work. F asserts the cost of implementing this defense–DAL-score pair as a lower bound for the defense in its corresponding component. Therefore, the constraints from F are necessary and sufficient. Given the minimization command to the MaxSMT solver, it will find a least cost model.

Case 2(a) adds upper bounds from currently satisfying implementations, so it still allows for a minimal cost model, with the restriction that new defenses may not be added.

Similarly, Case 2(b) adds lower bounds for already implemented defenses since we consider them a sunk cost, and minimizes given this restriction.

We have argued that the problem instance is soundly translated to MaxSMT, and that we can trust its result. We finish by arguing that our cost-inverse function soundly extracts the correct defense implementations for a minimal cost solution. This follows from the fact that our cost model is monotonically increasing, and that the cost-inverse function breaks the only possible ties from the costs by preferring higher DAL- scores when the cost of implementing a defense to a higher DAL-score is the same as that of implementing it to a lower DAL score (Section 4.3).