Attribute Repair for Threat Prevention*

Thorsten Tarrach[1], Masoud Ebrahimi[2], Sandra König[1], Christoph
Schmittner1, Roderick Bloem2, and Dejan Ničković[1]

[1] AIT Austrian Institute of Technology
[2] Graz University of Technology

Abstract. We propose a model-based procedure for preventing security
threats using formal models. We encode system models and threats as
satisfiability modulo theory (SMT) formulas. This model allows us to ask
security questions as satisfiability queries. We formulate threat preven-
tion as an optimization problem over the same formulas. The outcome
of our threat prevention procedure is a suggestion of model attribute
repair that eliminates threats. We implement our approach using the
state-of-the-art Z3 SMT solver and interface it with the threat analysis
tool THREATGET. We demonstrate the value of our procedure in two
case studies from automotive and smart home domains.

1 Introduction

The proliferation of communication-based technologies requires engineers to have
cybersecurity in mind when designing new applications. Historically, security
decisions in the early stages of development have been made informally. The
upcoming requirements regarding security compliance in soon-to-be-mandatory
standards such as the ISO/SAE 21434 call for more principled security assess-
ment of designs and the need for systematic reasoning about system security
properties has resulted in threat modeling and analysis tools. One example of
this new perspective is the Microsoft Threat Modelling Tool (MTMT) [8], devel-
oped as part of the Security Development Lifecycle. MTMT provides capabilities
for visual system structure modelling. Another example is THREATGET [12,3],
a threat analysis and risk management tool, originally developed in academia
and following its success, commercialized and used today by leading word-wide
companies in automotive and Internet-of-Things (IoT) domains. Threat mod-
eling and analysis significantly reduces the difficulty of a security assessment,
reducing it to accurate modeling of the systems and the security requirements.

Existing methods use ad-hoc methods to reason about the security of systems. As a result, it is not easy to extend such tools with model repair capabilities. Although a trial-and-error method is always possible, it does not provide a systematic exploration of the space of possible prevention measures and leaves the question of optimizing the cost of the prevention to the designer's intuition. As a result, remedying a potential threat remains cumbersome and simple solutions may be missed, especially in presence of multiple interacting threats.

This paper proposes a procedure for preventing threats based on a formal model of the structure of the system and a logic-based language for specifying threats. The use of rigorous, formal languages to model the system and specify threats allows us to automate threat prevention. More specifically, we reduce the problem of checking presence of threats in the system model to a satisfiability modulo theory (SMT) check. A threat specification defines a class of potential threats and a witness of a system model that satisfies a threat specification defines a concrete threat in the model. This allows us to frame the problem of preventing concrete threats as an attribute parameter repair.

The attributes of system components define a large spectrum of security settings and, in presence of a threat, of possible preventive actions. This class of repairs enables simple and localized measures whose cost can easily be assessed by a designer. We formulate attribute repair as a weighted maximum satisfiability (MaxSAT) problem with a model of cost of individual changes to the system attributes. This formulation of the problem allows us to find changes in the model with *minimal* cost that result in removing as many threats as possible.

We introduce *threat logic* as a specification language to specify threats. We formalize the system model as a logic formula that consists of a conjunction of sub-formulas, called *assertions*, parameterized by attributes that specify security choices. The conjunction of the system model formula and a negated threat formula is satisfiable iff there are no threats in the system. We introduce clauses that change the specific instantiation of model attributes to a different value and associate a *weight* with each such assertion. Then, the MaxSAT solution of this formula is the set of changes to system model attributes with minimum cost that ensure the absence of the threat. Given an incorrect system, we can choose the weights so that we compute the set of changes to system model attributes with the minimal cost to remove the existing threats from the model. To ensure that our method scales to industrial size models, we also define a heuristic that provides partial threat prevention by addressing repairable threats and explaining the reason why the others cannot be repaired. We believe that this method, even though partial and approximate in general, can compute near optimal repairs for many real-world problems.

We implemented the threat prevention method in the THREATGET tool and evaluated it on two case studies from the automotive and the IoT domains.

Motivating Example We motivate this work with a smart home application from the IoT domain, depicted in Figure 1. The smart home architecture consists of 7 *typed elements*: (1) a control system, (2) an IoT field gateway, (3)

temperature and (4) motion sensors, (5) a firewall, (6) a web server and (7) a mobile phone. The elements are interconnected using *wired* and *wireless connectors*. The elements and connectors have associated sets of *attributes* that describe their configuration. For instance, every connector has attributes Encryption, Authentication and Authorization. The attribute Encryption can be assigned the values No, Yes and Strong. We associate to each attribute a *cost* of changing the attribute value, reflecting our assessment of how difficult it is to implement the change. In this example, the temperature and the motion sensor communicate wirelessly with the gateway. If the motion sensor detects a movement, the user is notified by phone. It is possible to override the behavior, e.g., the heating can be turned on remotely in case of late arrival. The web server protected by the firewall allows for access and information exchange from and to the smart home. The IoT sub-system protected by the firewall defines a *security boundary* called the IoT Device Zone. Communication should be confidential and encrypted outside the IoT Device, which is represented by the two associated *assets*.
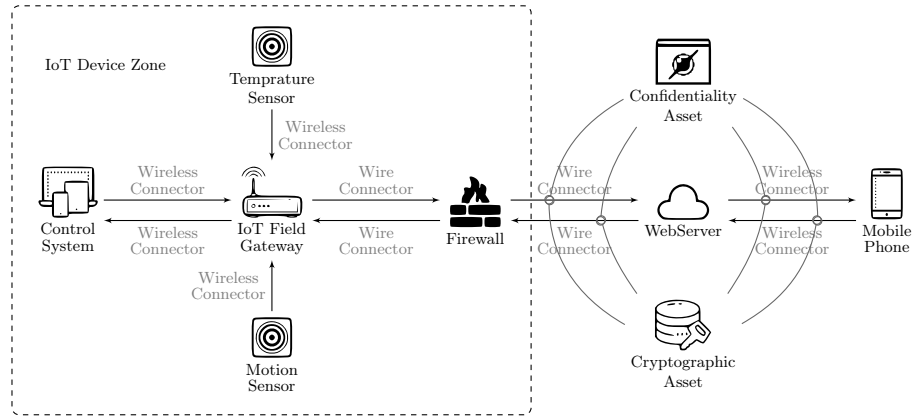

Fig. 1: Smart Home IoT model.

Threats in this smart home system are characterized by logical relations between elements, connectors and their attributes. Consider two potential threats that are applicable to this example: **Threat 1:** The web server enables data logging functionality without encrypting the data, and **Threat 2:** The mobile phone device is connected to the web server, without the web server enabling data logging. Assume that the web server has data logging enabled, but no data encryption, thus matching Threat 1. If we consider this threat in isolation, we can eithrt repair it by turning off the data logging, or by implementing the data encryption on the web server. The first repair results in matching Threat 2. Only the second repair results in the removal of all security threats. Given two data encryption algorithms with costs c_1 and c_2, where $c_1 > c_2$, implementing the latter is the cost-optimal option. We see that an optimal preventive solution must consider simultaneous repair of multiple threats.

2 Threat Modelling

A threat model consists of two main components, a *system model* and a database of *threat rules*. A system model provides an architectural view of the system under investigation, representing relevant components, their properties, as well as relations and dependencies between them [1].

System Model A system model M consists of:

- a set E of *elements*: an element $e \in E$ is a typed logical (software, database, etc.) or physical (ECUs, sensors, actuators, etc.) component.
- a set C of *connectors*: a connector $c \in C$ is a direct interaction between two elements, a *source* $\mathbf{s}(c) \in E$ and a *target* element $\mathbf{t}(c) \in E$.
- a set A of *security assets*: an asset $a \in A$ describes logical or physical object (such as an element or a connector) of value. Each element and connector can hold multiple assets. Similarly, each asset can be associated to multiple elements and connectors.
- a set B of *security boundaries*: a boundary $b \in B$ describes a separation between logically, physically, or legally separated system elements.
- a set $\mathfrak{A}$ of *attributes*: an attribute $\mathfrak{a} \in \mathfrak{A}$ is a property that can be associated to a system elements, connectors and/or assets. Each attribute $\mathfrak{a}$ can assume a value from its associated domain $D_\mathfrak{a}$. We denote by $v(x, \mathfrak{a})$ the value of the attribute $\mathfrak{a}$ associated to the element/connector/asset x. We finally define an *attribute cost* mapping $w_{x,\mathfrak{a}}(v, v')$ associated to $(x, \mathfrak{a})$ pairs that defines the cost of changing the attribute value $v \in D_\mathfrak{a}$ to $v' \in D_\mathfrak{a}$.

Given a system model M, we define a *path* π in M as an alternating sequence $e_1, c_1, e_2, c_2 \cdots, c_{n-1}, e_n$ of elements and connectors, such that for all $1 \le i \le n$, $e_i \in E$, for all $1 \le i < n$, $c_i \in C$, $\mathbf{s}(c_i) = e_i$, and $\mathbf{t}(c_i) = e_{i+1}$ and for all $1 \le i < j \le n$, $e_i \ne e_j$. We note that we define paths to be *acyclic*, since acyclic paths are sufficient to express all interesting security threats.

We use the notation elements(π) and connectors(π) to define the sets of all elements and of all connectors appearing in a path, respectively. The starting and the ending element in the path π are denoted by $e_{start}(\pi) = \mathbf{s}(c_1)$ and $e_{end}(\pi) = \mathbf{t}(c_{n-1})$, respectively. We denote by $P(M)$ the set of all paths in M.

Threat Logic We provide an intuitive introduction of *threat logic* for specifying potential threats [2]. The syntax of threat logic is defined as follows:

$$\varphi := R(X \cup P) \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \exists p.\varphi \mid \exists x.\varphi$$

where $X = E \cup C \cup A \cup B$, $x \in X$, P is a set of path variables, $p \in P$, and $R(X \cup P)$ is a predicate. The predicate $R(X \cup P)$ is of the form:

---

[1] The system and the threat model are formally defined in Appendices A and B.

[2] THREATGET uses its own syntax and semantics to express threats [3]. We use instead predicate logic to facilitate the encoding of the forthcoming algorithms into SMT formulas. Our implementation contains an automated translation from THREATGET syntax to threat logic.

1. $\mathsf{type}(x) = t$ - the type of $x \in X$ is t;
2. x in p - the element or the connector $x \in E \cup C$ is in the path $p \in P$;
3. $\mathsf{connector}(e, c)$ - the element $e \in E$ is either the source or the target of the connector $c \in C$;
4. $\mathsf{src}(c) = e$ - the source of the connector $c \in C$ is the element $e \in E$;
5. $\mathsf{tgt}(c) = e$ - the target of the connector $c \in C$ is the element $e \in E$;
6. $\mathsf{src}(p) = e$ - the source of the path $p \in P$ is the element $e \in E$;
7. $\mathsf{tgt}(p) = e$ - the target of the path $p \in P$ is the element $e \in E$;
8. $\mathsf{crosses}(c, b)$ - the connector $c \in C$ crosses the boundary $b \in B$;
9. $\mathsf{contained}(x, b)$ - the element or boundary $x \in E \cup B$ is contained in the boundary $b \in B$;
10. $\mathsf{holds}(x, a)$ - the element or the connector $x \in E \cup C$ holds the asset $a \in A$;
11. $\mathsf{val}(x, att) = v$ - the valuation of the attribute att associated to $x \in E \cup C \cup A$ is equal to v.

Example 1. Consider a requirement that there exists a path in the model such that all the elements in that path are of type Cloud. It is expressed with the threat logic formula: $\exists p. \forall e. (e\ \mathsf{in}\ p \implies \mathsf{type}(e) = \text{Cloud})$.

We define an *assignment* Π_M as a partial function that assigns element, connector, asset, security boundary and path variables to concrete elements, connectors, assets, security boundaries and paths from the system architecture model M. We denote by $\Pi_M[x \mapsto i]$ the item assignment in which x is mapped to i and otherwise identical to Π_M. Similarly, we denote by $\Pi_M[p \mapsto \pi]$ the path assignment in which p is mapped to π and otherwise identical to Π_M. The semantics of threat logic follow the usual definitions of predicate logic.

We say that a threat logic formula is *closed* when all occurrences of element, connector, asset and security boundary variables are in the scope of a quantifier. Any closed threat logic formula is a valid threat specification. Given a system model M and a closed threat logic formula φ, we say that M *witnesses* the threat φ, denoted by $M \models \varphi$ iff $\Pi_M \models \varphi$, where Π_M is an empty assignment.

From Threat Logic To First Order Logic (FOL) We interpret threat logic formulas over system models with a finite number of elements and connectors, and hence we can eliminate path quantifiers by enumerating the elements and connectors in the path. We thus obtain an equisatisfiable FOL formula that can be directly used by an SMT solver. The path quantifier elimination procedure T that takes as input a threat formula φ and computes the equisatisfiable FOL formula $T(\varphi)$ is formalized in Appendix 4.

Example 2. We formalize the two threats described in Section 1:

Threat 1 $\exists e. (\mathsf{type}(e) = \text{WebServer} \wedge \mathsf{val}(e, \text{Data Logging}) = \text{Yes}) \wedge \mathsf{val}(e, \text{Data Encryption}) \neq \text{Yes})$

Threat 2 $\exists p, e_1, e_2. \mathsf{src}(p) = e_1 \wedge \mathsf{tgt}(p) = e_2 \wedge$
$\mathsf{type}(e_2) = \text{WebServer} \wedge \mathsf{type}(e_1) = \text{MobilePhone} \wedge \mathsf{val}(e_2, \text{Data Logging}) \neq \text{Yes}$

3 Automated Threat Prevention

We now present our main contribution – a procedure to automatically repair a system model with one or more threats. We restrict our attention to the class of *attribute repairs* that consists in changing the model attribute values and show how to encode the problem using optimization modulo theories. We first present an exact algorithm for the minimal attribute repair using an SMT solver, and then propose a more scalable heuristic for partial repair.

3.1 Attribute Repair

In this work, repairing a model that has one or multiple threats consists in changing the attribute valuation function of the model. Not every model can be attribute repaired. For a threat model M we denote by $M[v'\backslash v]$ the threat model in which the attribute value assignment v is replaced by another assignment v'.

Definition 1 (Threat-repairable model). *Given a model M with an attribute valuation function v that witnesses a threat φ, $M \models \varphi$, we say that M is* attribute repairable *wrt φ iff there exists a v' such that $M[v'\backslash v] \not\models \varphi$.*

We specifically aim at finding the *optimal* repair, which has the minimal repair cost. To reason about this quantitative repair objective, Definition 2 specifies the *distance* $d(v, v')$ between two attribute valuation functions v and v' as the sum of altered attribute costs for attributes that differ in the two valuations.

Definition 2 (Attribute valuation distance). *Let M be a system model with attribute valuation v and attribute cost w. Let v' be another attribute valuation. The* distance $d(v, v')$ *between v and v' is defined as:*

$$d(v, v') = \sum_{x \in X, \mathfrak{a} \in \mathfrak{A}} w_{x,\mathfrak{a}}(v(i, \mathfrak{a}), v'(x, \mathfrak{a})) \ \ s.t. \ v(x, \mathfrak{a}) \neq v'(x, \mathfrak{a}).$$

For instance, for the attribute 'Encryption' the cost of changing from 'None' to 'Weak' may be 20, but to change 'None' to 'Strong' may cost 30. A change from 'Weak' to 'Strong' could cost 15, but a change from 'Weak' to 'None' may only cost 1. Sensible cost functions will adhere to some restrictions (such as a variant of the triangle inequality) that we do not formalize here.

Definition 3 (Minimal attribute repair). *Let φ be a threat logic formula and M a system model such that $M \models \varphi$ and M is attribute repairable w.r.t. φ. The* minimal attribute repair *of M is another threat model $M[v'\backslash v]$ s.t.:*

$$v' = \arg\min\{d(v, v'')|M[v''\backslash v] \not\models \varphi\}$$

Other notions of minimal repair. There are at least two other natural notions of minimal repair. In the first one, costs are associated with the attribute itself. This means that every change of the attribute carries the same cost. We can model this by assigning the same cost to all possible combinations of previous

and new value for an attribute. Alternatively, engineers are often not interested in minimizing the overall real cost, but rather in minimizing the number of attributes that need to be repaired. We can model this restricted variant of the problem by associating the fixed cost of 1 each attribute in the model, thus effectively counting the number of individual attribute repairs. Both variants can be implemented in a straightforward manner in our framework.

3.2 Attribute Repair as Weighted MaxSMT

We encode the attribute repair problem (see Section 3.1) as a *weighted MaxSMT* problem, in which $\mathbf{F}$ represents a (hard) *assertion*, while $F_1, \ldots, F_m$ correspond to *soft assertions* and every soft assertion F_i has an associated cost $cost_i$.

Definition 4 (Weighted MaxSMT [1]). *Given an SMT formula $\mathbf{F}$, a set of SMT formulas $F_1, \ldots, F_m$ and a set of real-valued costs $cost_1, \ldots, cost_m$, the weighted MaxSMT problem consists in finding a subset $K \subseteq M$ of indices with $M = \{1, \ldots, m\}$ such that: (1) $\mathbf{F} \wedge \bigwedge_{k \in K} F_k$ is satisfiable, and (2) the total cost $\sum_{i \in N \setminus K} cost_i$ is minimized.*

We now sketch the encoding of the minimal attribute repair problem into weighted MaxSMT. We assume that we have a MaxSMT solver object, with the following functionality: $push()$ - push new context to solver stack, $pop()$ - pop context from solver stack, $add(\varphi)$ - add new hard assertion φ, $add\_soft(\varphi, c)$ - add new soft assertion φ with weight c, $solve()$ - check if formula is satisfiable, $max\_solver()$ - check if formula is max-satisfiable, and $model()$ - generate and return a model witnessing satisfaction of a formula.

Given a system model M and a set of threat logic formulas $\Phi = \{\varphi_1, \ldots, \varphi_n\}$, we compute the MaxSMT formulas $\mathbf{F}$ that represents the hard assertion $\mathbf{F} = F_M \wedge \bigwedge_{j=1}^{n} \neg\varphi_j$ conjoins F_M that encodes the entire system model *except* its attribute valuations and costs with the negation of each threat logic formula φ_j. We also define one soft assertion $F_{x,\mathfrak{a},v}$ for each element x, attribute $\mathfrak{a}$ of x and possible value v of $\mathfrak{a}$, stating intuitively that $v(x, \mathfrak{a}) = v$. These soft attributes are mutually exclusive if they assert different values for the same attribute. We set up the costs of each $F_{x,\mathfrak{a},v}$ in such a way that asserting $F_{x,\mathfrak{a},v}$ leads to cost corresponding to changing the value of $\mathfrak{a}$ to v. (The exact value of the cost function can easily be computed by solving a linear system of equations.)

We use the weighted MaxSAT solver $\max\,\mathrm{solve}(F_M \wedge \bigwedge_{j=1}^{n} \neg\varphi_j \wedge \bigwedge F_{x,\mathfrak{a},v})$ to obtain the satisfiability verdict and the optimization cost. Informally, the solver can return three possible verdicts:[3]

- sat verdict with total cost 0: the system model M does not contain a potential threat defined by any of the threat formulas φ_i,
- sat verdict with total cost k: the system model M contains a set of potential threats defined by a subset of threat formulas and can be repaired by

---

[3] We ignore here a fourth possible verdict unknown that can arise in practice and that happens if the solver is not able to reach a conclusion before it times out.

changing the values of model attributes with total cost k. The solver returns a model, which defines a possible repair, i.e. the altered attribute values that render the formula satisfiable,

– unsat verdict: the system model M is not attribute repairable with respect to at least one threat formula φ_i.

The encoding of the attribute repair into this MaxSMT problem provides an effective solution to the minimum attribute repair problem.

Theorem 1. *Let M be a system model and $\{\varphi_1, \ldots, \varphi_n\}$ a set of closed threat logic formula. We have that $\max \mathrm{solve}(F_M \wedge \bigwedge_{i=1}^{n} \neg \varphi_i \wedge \bigwedge_{F \in \Psi} F)$ provides the solution to the minimum attribute repair problem.*

3.3 Partial Repair of Unrepairable Models

The problem with the approach from Section 3.2 arises if there is a formula φ_i for which M is not attribute repairable. In that case, the entire problem is unsatisfiable, even if other threats could be repaired. This outcome, although correct, is not of particular value to the security engineer. Ideally, the objective is to repair attributes for threats that can be repaired and explain the others.

We observe that attribute-unrepairable threats have a particular form and correspond to formulas without constraints on attribute valuations. An inductive visit of the formula allows a syntactic check has_attr(φ) whether a threat formula φ has any constraint on attribute valuations. Algorithm 1 implements PartialRepair, a method that adapts the MaxSMT algorithm from Section 3.2 to compute a partial repair of M with respect to a subset of repairable threat formulas. The procedure removes all threat logic formulas that are satisfied by the model and that are known to be unrepairable, before computing MaxSMT.

From the definition of the partial repair algorithm, it follows that the MaxSAT applied to the subset of (potentially) repairable threat formulas corresponds to the minimum attribute repair restricted to that subset of threat formulas.

Corollary 1. *Let M be a system model, $\Phi = \{\varphi_1, \ldots, \varphi_n\}$ a set of closed threat logic formulas and $G \subseteq \Phi$ a subset of repairable threats, i.e. for all $\varphi \in G$, $M \not\models \varphi$ or has attr(φ) is true. We have that $\max \mathrm{solve}((F_M \wedge \bigwedge_{\varphi \in G} \neg \varphi) \wedge \bigwedge_{F \in \Psi} F)$ provides the solution to the minimum attribute repair problem restricted to the set G of threat formulas.*

Explaining unrepairable threats: The partial repair method is useful in the presence of threats that cannot be addressed by attribute repair only. The SMT solver can be used to provide in addition an explanation of why a threat cannot be repaired – the solver assigns values to variables in threat logic formulas that witness its satisfaction (i.e. the presence of that threat). This witness explains exactly why that formula is satisfied and locates in the system model the one set of items that are responsible for that verdict. The threat may match at multiple locations of the model, which could be discovered by multiple invocations of the solver (while excluding the previously found sets). Note that the MaxSMT

will repair all occurrences though, because the threat is negated there and the negated existential quantifier becomes a forall quantifier.

The procedure takes as input the solver (we assume that it already has the system model encoding that includes all hard assertions), the set of attribute valuations given in the form of assertions, and the set of threat logic formulas. The procedure maintains two sets of threat logic formulas, the ones that do not pose any threat to the system model and the ones that do pose a threat, but are repairable (line 1). The set of unrepairable formulas are implicitly the ones that are in neither of these two sets.

Algorithm 1 Partial attribute repair: PartialRepair()

Input : $M, \{\varphi_1, \ldots, \varphi_n\}, F_M, \hat{F}_M, \Psi$
Output: status of repair, cost of repair,
            list of repaired attribute values $\hat{\Psi}$

1 solver $\leftarrow$ SMTSolver() ;
2 solver. add($\hat{F}_M$) ;
3 $G \leftarrow \emptyset$;
4 **for** $\varphi \in \{\varphi_1, \ldots, \varphi_n\}$ **do**
5   solver. push() ;
6   solver. add($T(\varphi)$) ;
7       $status \leftarrow$ solver. solve() ;
8   solver. pop() ;
9   **if** $status = $ sat **then**
10   **if** has attr(φ) **then**
11              $G \leftarrow G \cup \{\varphi\}$;
12   **else if** $status = $ unsat **then**
13          $G \leftarrow G \cup \{\varphi\}$;

14 solver $\leftarrow$ SMTMaxSolver() ;
15 solver. add(F_M) ;
16 **for** $F \in \Psi$ **do**
17   solver. add soft($F, cost(F)$) ;
18 **for** $\varphi \in G$ **do**
19   solver. add($\neg T(\varphi)$) ;
20 $status \leftarrow$ solver. max solve() ;
21 $\hat{\Psi} \leftarrow \emptyset$;
22 $totalcost \leftarrow 0$;
23 **if** $status = $ sat **then**
24       $m \leftarrow$ solver. model() ;
25       $\hat{\Psi}, totalcost \leftarrow$ repair(m, Ψ) ;
26 **return** $status, totalcost, \hat{\Psi}$

For every threat logic formula φ, the procedure first checks if that threat is present in the system model using the SMT solver (lines 5–10). If the threat is absent, it is added to the set of formulas that do not represent any threat (lines 11–12). Otherwise, the procedure attempts to compute a repair for that particular threat (lines 13–33). It first checks whether the threat logic formula refers to any attribute valuations (line 14). If not, the formula is unrepairable. Otherwise (lines 15–19), the formula is added as a hard assertion, and the set of attribute valuations are added as soft assertions are added as soft assertions to the solver, and the MaxSMT solver is invoked. If the solver gives unsat verdict, it means that the model cannot be repaired to satisfy the threat logic formula. Otherwise (lines 20–32), we use the model witnessing the satisfaction of the threat logic formula to compute the partial repair for that formula (line 22). The outcome of the repair method is a the repaired set of attribute assertions and the number of assertions that needed to be altered. The procedure checks that this partial repair is consistent with the previous repairs (i.e. that it does not lead to violation of other previously processed threat logic formulas), and the repair is accepted only upon passing this last consistency check (lines 24-32).

Removing unrepairable formulas in Algorithm 1 does not guarantee that the remaining threat formulas can be simultaneously repaired, e.g. in presence

mutually inconsistent threat formulas. We hence propose a heuristic procedure for partial repair in Algorithm 2.

4   Implementation and Case Studies

We did a prototype implementation of the proposed methods in Java and integrated it to THREATGET. We used THREATGET's threat database from automotive and IoT domains originating from multiple sources, including security-related standards and previously discovered threats. The tool imports system models as JSON files and threat descriptions as THREATGET rules, translates both to FOL SMT formulas and uses Z3 as the MaxSMT solver. The MaxSMT solver's results are used to compute the repair suggestions. While THREATGET is a proprietary software (with a free academic license), our threat repair extension is distributed under the BSD-3 license. We assume the default attribute change cost 1 and allow the user to change it using a CSV file. We apply our tool to two case studies from two domains: the smart home IoT application introduced as our motivating example in Section 1, and the vehicular telematic gateway. A third key fob case study can be found in Appendix C, in which we compare the vanilla MaxSMT approach to the heuristic procedure.

Algorithm 2 Approximate partial repair: HPartialRepair

Input : M, $\{\varphi_1, \ldots, \varphi_n\}, F_M, \Psi$

Output: Repair status and cost, set of repaired and non-repaired threats, repaired set of attributes

1 $nothreat \leftarrow \emptyset$; $repairable \leftarrow \emptyset$; $totalcost \leftarrow 0$;

2 solver $\leftarrow$ SMTMaxSolver() ;

3 solver. add(F_M) ;

4 for $\varphi \in \{\varphi_1, \ldots, \varphi_n\}$ **do**

5 $\quad$ solver. push() ;

6 $\quad$ **for** $F \in \Psi$ **do**

7 $\quad\quad$ solver. add(F) ;

8 $\quad$ solver. add($T(\varphi)$) ;

9 $\quad$ $status \leftarrow$ solver. solve() ;

10 $\quad$ solver. pop() ;

11 $\quad$ **if** $status =$ unsat **then**

12 $\quad\quad$ $nothreat \leftarrow nothreat \cup \{\varphi\}$;

13 $\quad$ **else if** $status =$ sat **then**

14 $\quad\quad$ **if** has attr(φ) **then**

15 $\quad\quad\quad$ solver. push() ;

16 $\quad\quad\quad$ **for** $F \in \Psi$ **do**

17 $\quad\quad\quad\quad$ solver. add soft($F, cost(F)$) ;

18 $\quad\quad\quad$ solver. add($\neg T(\varphi)$) ;

19 $\quad\quad\quad$ $status \leftarrow$ solver. max solve() ;

20 $\quad\quad\quad$ **if** $status =$ sat **then**

21 $\quad\quad\quad\quad$ $m =$ solver. model() ;

22 $\quad\quad\quad\quad$ $\hat{\Psi}, c \leftarrow$ repair(m, Ψ) ;

23 $\quad\quad\quad\quad$ solver. pop() ;

24 $\quad\quad\quad\quad$ solver. push() ;

25 $\quad\quad\quad\quad$ solver. add($\bigvee_{\varphi' \in repairable \cup nothreat} T(\varphi')$) ;

26 $\quad\quad\quad\quad$ **for** $F \in \hat{\Psi}$ **do**

27 $\quad\quad\quad\quad\quad$ solver. add(F) ;

28 $\quad\quad\quad\quad$ $status \leftarrow$ solver. solve() ;

29 $\quad\quad\quad\quad$ **if** $status =$ unsat **then**

30 $\quad\quad\quad\quad\quad$ $repairable \leftarrow repairable \cup \{\varphi\}$;

31 $\quad\quad\quad\quad\quad$ $\Psi \leftarrow \hat{\Psi}$;

32 $\quad\quad\quad\quad\quad$ $totalcost \leftarrow totalcost + c$;

33 $\quad\quad\quad$ solver. pop() ;

34 return $status, totalcost, repairable, nothreat, \Psi$

Smart Home IoT Application This case study was introduced in Section 1. In this section, we report on the experimental results obtained by applying our threat repair approach. The model has been analysed against 169 IoT-related threat descriptions given in the form of threat log formulas. We applied both the full MaxSMT optimization procedure and its heuristic variant.

Both the model and the database of IoT threat formulas are publicly available. Table 1 summarizes the outcomes. To accurately report the number of repairable formulas we implement Algorithm 2 without line 31. The cost reflects

the number of attributes (per item) changed in the model because we set the cost per attribute to 1 in our experiments. We can observe that the heuristic procedure was able to repair 27 out of 36 found threats in less than 50 seconds.

Table 1: Results of attribute repair applied to Smart Home IoT case study.

verdict	SAT	# formulas w/t threat	133
total # formulas	169	total cost	77
# repairable formulas	27	time (s)	46.7
# unrepairable formulas	9		

We illustrate the repair process on two threats from the database of IoT security threats. We first consider the threat with the title "Attacker can deny the malicious act and remove the attack foot prints leading to repudiation issues". This threat is formalized using the threat logic formula

$$\exists e. \ \mathsf{type}(e) = \text{Firewall} \ \wedge (v(e, \text{Activity Logging}) \neq \text{Yes} \ \vee v(e, \text{Activity Logging}) = \text{Missing}).$$

The SMT solver finds that the model satisfies the above formula and hence has a threat. The witness shows that one element of type 'Firewall' has the 'Activity Logging' attribute set to 'undefined', thus explaining the threat. The proposed repair consists in implementing the activity logging functionality. This repair was found to be consistent with the other ones and is reported as part of the overall repair suggestion. The second threat has the title "Spoofing IP" and is reported as an irreparable threat. It is formalized using the threat logic formula

$$\exists c. \exists e_1. \exists e_2 \ \mathsf{type}(c) = \text{Internet Connection} \ \wedge \mathsf{src}(c) = e_1 \wedge \mathsf{tgt}(c) = e_2.$$

This threat cannot be repaired by changing the model attributes. On the contrary, this threat formula states that any connection to the internet constitutes a potential IP spoofing threat.

Vehicular telematic gateway This study is based on an industrial-strength model of a vehicular telematic gateway (VTG) [9]. The VTG connect internal elements of the vehicle with external services. it offers vehicle configuration and entertainment and navigation to the user. An item is a term introduced by ISO 26262, describing a system or combination of systems, enabling a function on the vehicle level. The industrial company that devised the THREATGET model develops such systems for usage by different vehicle manufacturer so that the item is developed based on assumptions about the vehicle that need validation.

We consider a telematic ECU that offers remote connectivity for the on-board network to support various remote services including data acquisition, remote control, maintenance, and over-the-air (OTA) software update. It also provides a human-machine interface (HMI) for navigation, configuration, and multimedia control. HMI represents the central element in a vehicle, connecting the control system to the human operator and the backend. It has cellular and

wireless local area network (WLAN) interfaces for wireless connectivity. For local connectivity, it includes a USB port for software updates and application provision. The telematics system is also connected to other onboard ECUs.

For our analysis we considered two variants of time-triggered control. A simplified model contains 15 elements, 24 transitions and 5 assets, while the full model has 25 elements, 48 transitions and 5 assets. The presence of two models reflects the iterative design process in which the high-level simplified model was refined into the complete model based on the previous analysis.

A model of the full version is shown in Figure 3 (Appendix D). The results of the attribute repair are presented in Table 2. We see that the tool is able to scale to large industrial models. It analyses 95 threat formulas in 497s, repairing 19 out of 42 threats with the cost 57. The same set of rules were repaired in 127s on the simplified model. The main complexity in repairing large models comes from the threat formulas that contain quantification over path. This is not surprising because each such formula corresponds to a bounded model checking (reachability) problem. To confirm this observation, we analysed the full model without formulas with quantification over paths. The analysis of 82 such formulas was done in less than 118s, resulting in the repair of 18 out of 39 threats.

Table 2: Results of attribute repair for two vehicular telematic gateways.

	with flow		w/o flow
	simple	full	full
verdict	SAT	SAT	SAT
total # formulas	95	95	82
# repairable formulas	15	19	18
# unrepairable formulas	25	23	21
# formulas w/t threat	55	53	43
total cost	29	57	57
time (s)	127	497	118

One repaired threat was *Spoofing sensors by external effects*. It was present because a CAN interface is connected to an ECU (the secondary CPU) and they both hold an asset (the Communication Interface). It represents the possibility that the assets could be attacked to send incorrect data to vehicle sensors (i.e., radar signals). That could lead to giving incorrect decisions based on the tampered input signal, affect the safe operation of the vehicle, or impact on usual vehicle functionalities. Our tool suggests to this threat by implementing an input validation on the Secondary CPU, which enables the CPU to detect false data.

5 Related work

Threat modeling and analysis has received increasing interest in the recent years, both in academia and industry. A plethora of commercial and open-source threat modeling tools have been developed, including THREATGET [12,3], Microsoft Threat Modeling Tool [8], ThreatModeler [15], OWASP Threat Dragon [7]

and pytm [17], Foreseeti [6], Security Compass SD Elements [14] and Tutamen Threat Model Automator [16]. These tools can be divided into three categories: (1) manual approaches based on excel sheets or questionnaires [16], (2) graphical modelling approaches *without* an underlying formal model [8,7,15,14], and (3) model-based system engineering tools *with* an underlying formal model [6,17,12,3]. The first class of tools does not admit automated threat analysis and any threat prevention measure must be manually identified and selected. Several tools from the second and third class [8,6,12,3] provide some limited form of hard-coded measures that are associated to individual threats or assets, without considering threat inter-dependencies nor mitigation costs and are not able to compute a global and consistent set of threat prevention measures. Hence our threat prevention approach could be integrated to theses classes of tools.

Optimization Modulo Theories (OMT) combine SMT solvers with optimization procedures [2,4,5,10,13]. to find solutions optimizing some objective function Parameter synthesis using SMT solvers does not require optimization objectives in general. Bloem et al. [11] synthesized parameter values ensuring safe behavior of cyber-physical systems through solving an $\exists\forall$ SMT formula. In this work, users specify safe states in terms of state and parameter values; then, the synthesizer attempts to compute correct parameter values conforming to an invariant template such that for all possible inputs all reachable states are safe.

6   Discussion and Future Work

We presented a framework that enables automated threat prevention by repairing security-related system attributes. Although widely applicable, attribute-value repair is not enough to cover all interesting preventive procedures. For example, protecting safety-critical components connected to a Controller Area Network (CAN) bus in a vehicle cannot be done just by encrypting messages. In fact, encryption is not part of the CAN protocol. The preventive measure would require separating trusted (safety-critical) part of the system from the untrusted one (entertainment system, etc.) with a firewall, a measure that is beyond the attribute-value repair. Despite few similar examples, the attribute repair remains a suitable repair strategy for the majority of threats present in common architectures. Intuitively, this is the case because the attributes document the counter-measures taken against common classes of threats, e.g. authentication as a counter-measure against escalation of privilege. The more general *model repair* problem can addresses the limitations illustrated by the CAN example by enabling addition and removal of elements in the model. Unrestricted alteration of models would lead to trivial and uninteresting repairs, e.g. it suffices to disconnect all elements in the model from each other to disable the vast majority of threats. Hence, model repair requires a restriction of repair operations and even just identifying a set of useful repair operations is a challenging task. The more general model repair is a separate research problem that differs in several key aspects from the attribute-value repair and we plan to tackle it as future work.

References

1. N. Bjørner and A. Phan. νz - maximal satisfaction with Z3. In *6th International Symposium on Symbolic Computation in Software Science, SCSS 2014*, pages 1–9, Gammarth, La Marsa, Tunisia, Dec. 2014. EasyChair.

2. N. Bjørner, A. Phan, and L. Fleckenstein. νZ - An Optimizing SMT Solver. In C. Baier and C. Tinelli, editors, *TACAS 2015*, volume 9035 of *Lecture Notes in Computer Science*, pages 194–199, London, UK, April 2015. Springer.

3. K. Christl and T. Tarrach. The analysis approach of threatget. *CoRR*, abs/2107.09986, 2021.

4. A. Cimatti, A. Griggio, B. J. Schaafsma, and R. Sebastiani. A modular approach to maxsat modulo theories. In M. Järvisalo and A. V. Gelder, editors, *SAT 2013*, volume 7962 of *Lecture Notes in Computer Science*, pages 150–165, Helsinki, Finland, July 2013. Springer.

5. I. Dillig, T. Dillig, K. L. McMillan, and A. Aiken. Minimum satisfying assignments for SMT. In P. Madhusudan and S. A. Seshia, editors, *CAV 2012*, volume 7358 of *Lecture Notes in Computer Science*, pages 394–409, Berkeley, CA, USA, July 2012. Springer.

6. Foreseeti AB. Foreseeti. Online, 2020. Accessed: 2020-11-29.

7. M. Goodwin and J. Gadsden. Owasp threat dragon. Online, 2020. Accessed: 2020-11-29.

8. R. McRee. Microsoft threat modeling tool 2014: identify & mitigate. *ISSA Journal*, 39:42, 2014.

9. M. W. Mürling. Security by design: New "THREATGET" tool tests cyber security in vehicles and systems. Online Article, 2021.

10. R. Nieuwenhuis and A. Oliveras. On SAT modulo theories and optimization problems. In A. Biere and C. P. Gomes, editors, *SAT 2006*, volume 4121 of *Lecture Notes in Computer Science*, pages 156–169, Seattle, WA, USA, Aug. 2006. Springer.

11. H. Riener, R. Könighofer, G. Fey, and R. Bloem. SMT-based CPS parameter synthesis. In G. Frehse and M. Althoff, editors, *ARCH@CPSWeek 2016, 3rd International Workshop on Applied Verification for Continuous and Hybrid Systems*, volume 43 of *EPiC Series in Computing*, pages 126–133, Vienna, Austria, 2016. EasyChair.

12. M. E. Sadany, C. Schmittner, and W. Kastner. Assuring compliance with protection profiles with threatget. In *SAFECOMP 2019 Workshops*, Lecture Notes in Computer Science, pages 62–73, Berlin, 2019. Springer.

13. R. Sebastiani and P. Trentin. Optimathsat: A tool for optimization modulo theories. *J. Autom. Reason.*, 64(3):423–460, 2020.

14. Security Compass Ltd. Security compass sd elements. Online, 2020. Accessed: 2020-11-29.

15. ThreatModeler Software, Inc. Threatmodeler. Online, 2020. Accessed: 2020-11-29.

16. Tutamantic Ltd. Tutamen threat model automator. Online, 2020. Accessed: 2020-11-29.

17. J. Was, P. Avhad, M. Coles, N. Ozmore, R. Shambhuni, and I. Tarandach. Owasp pytm. Online, 2020. Accessed: 2020-11-29.

A   System Model

This section defines a formal model that we use to analyze a system against a database of threats. A system model provides an architectural view of the system under investigation, representing relevant components, their properties, as well as relations and dependencies between them. Complete, formal definitions of the system and the threat model are given in [3].

The system model is designed from a set of available entities, such as *elements*, *connectors*, *security assets* and *security boundaries.* These entities are typed, can have attributes (properties) assigned to values drawn from some domain. The entity types, attributes and domains are not arbitrary – they are selected from a predefined "arena" that we call a *meta model.*

Before providing a formal definition of a meta model, we introduce useful notation. We use the term *item* to refer to a generic model entity when distinction is unnecessary. In particular, we denote by $\mathcal{I} = \mathcal{E} \cup \mathcal{C} \cup \mathcal{A} \cup \mathcal{B}$ the universe of *items*, where $\mathcal{E}$, $\mathcal{C}$, $\mathcal{A}$ and $\mathcal{B}$ are disjoint sets of all possible *elements*, *connectors*, *assets* and *boundaries*, respectively.

In our framework, every item has a *type.* We denote by $\mathbb{E}$, $\mathbb{C}$, $\mathbb{A}$ and $\mathbb{B}$ the sets of valid types for elements, connectors, assets and boundaries, resp. We also use the notation $\mathbb{I} = \mathbb{E} \cup \mathbb{C} \cup \mathbb{A} \cup \mathbb{B}$ to group the types for generic items.

All items have *attributes* and attributes are assigned *values.* Let $\mathfrak{A}$ denote the set of all attributes, and $\mathfrak{V}$ the set of all attribute values. The mapping $\mathfrak{D} : \mathfrak{A} \to 2^{\mathfrak{V}}$ defines the *domain* $\mathfrak{D}(\mathfrak{a}) \subseteq \mathfrak{V}$ of the attribute $\mathfrak{a} \in \mathfrak{A}$.

Definition 5 (Meta model). *A* meta model $\mathcal{M}$ *is a tuple* $\mathcal{M} = (\mathbb{I}, \mathfrak{A}, \mathfrak{V}, \mathfrak{D}, \mathbf{a})$, *where*

 – $\mathbb{I}$, $\mathfrak{A}$ *and* $\mathfrak{V}$ *are finite sets of* item types, attributes *and* attribute values,
 – $\mathfrak{D}$ *is a function mapping attributes to sets of values, and*
 – $\mathbf{a} : \mathbb{I} \to 2^{\mathfrak{A}}$ *is an* attribute labelling *that maps every item type* $i \in \mathbb{I}$ *to a finite set* $\mathbf{a}(i) \subseteq \mathfrak{A}$ *of attributes.*

We are now ready to define a *system model* M, which is instantiated by entities selected from its associated meta model $\mathcal{M}$. Definition 6 formalizes a system model.

Definition 6 (System Model). *A* system model M *is a tuple* $M = (\mathcal{M}, I, \tau, \boldsymbol{s}, \boldsymbol{t}, v, w, \beta, \alpha)$, *where:*

 – $\mathcal{M}$ *is a meta model,*
 – $I \subseteq \mathcal{I}$ *is a finite set of* items, *partitioned into a disjoint union of* elements $E \subseteq \mathcal{E}$, connectors $C \subseteq \mathcal{C}$, assets $A \subseteq \mathcal{A}$, *and* boundaries $B \subseteq \mathcal{B}$,
 – $\tau : I \to \mathbb{I}$ *is the* type labelling *function that maps items* $i \in I$ *to their types* $\tau(i) \in \mathbb{I}$, *such that (1) if* $i \in E$, *then* $\tau(i) \in \mathbb{E}$, *(2) if* $i \in C$, *then* $\tau(i) \in \mathbb{C}$, *(3) if* $i \in A$ *then* $\tau(i) \in \mathbb{A}$, *and (4) if* $i \in B$ *then* $\tau(i) \in \mathbb{B}$,
 – $\boldsymbol{s} : C \to E$ *and* $\boldsymbol{t} : C \to E$ *denote functions that map a connector to its* source *and* target *elements, respectively.*

- $v : I \times \mathfrak{A} \rightharpoonup \mathfrak{V}$ *is a* partial attribute valuation *function, where (1)* $v(i, \mathfrak{a})$ *is defined if* $\mathfrak{a} \in \mathbf{a}(\tau(i))$*, and (2) if* $v(i, \mathfrak{a})$ *is defined, then* $v(i, \mathfrak{a}) \in \mathfrak{D}(\mathfrak{a})$*.*
- $w : I \times \mathfrak{A} \times \mathfrak{D} \times \mathfrak{D} \rightharpoonup \mathbb{R}_{\geq 0}$ *is a* partial attribute cost *function such that* $w(i, \mathfrak{a}, x, x')$ *is defined only if* $v(i, \mathfrak{a})$ *is defined and* $x, x' \in \mathfrak{D}(\mathfrak{a})$*, which denotes the cost of changing the attribute* $\mathfrak{a}$ *from value* x *to* x' *for item* i*.*
- $\beta \subseteq B \times (B \cup E)$ *is a* binary boundary containment *relation that encodes a tree in which the leaves are elements of* E *and the internal nodes are elements of* B*. We use* β^* *to denote the transitive closure of the boundary containment relation.*
- $\alpha : (E \cup C) \times A$ *is the* asset relation *between assets on one side, and elements and connectors on the other side. Each element/connector can hold multiple assets. Similarly, each asset can be associated to multiple elements and connectors.*

Definition 7 (Path). *Given a system model* M*, a path in* M *is an alternating sequence of elements and connectors in the form of* $\pi = e_1, c_1, e_2, c_2 \cdots, c_{n-1}, e_n$*, such that for all* $1 \leq i \leq n$*,* $e_i \in E$*, for all* $1 \leq i < n$*,* $c_i \in C$*,* $\mathbf{s}(c_i) = e_i$*, and* $\mathbf{t}(c_i) = e_{i+1}$ *and for all* $1 \leq i < j \leq n$*,* $e_i \neq e_j$*.* [4]

We use the notation elements(π) and connectors(π) to define the sets of all elements and of all connectors appearing in a path, respectively. The starting and the ending element in the path π are denoted by $e_{start}(\pi) = \mathbf{s}(c_1)$ and $e_{end}(\pi) = \mathbf{t}(c_{n-1})$, respectively. We denote by $P(M)$ the set of all paths in M.

Example 3. We illustrate the system model on the generic IoT example from Section 1. The model consists of 4 elements (Cloud, Fog Node, IoT Field Gateway, Intelligent Monitoring Device) and 6 connectors. Table 3 associates elements to their respective types, drawn from the meta model. There is only one type, Communication Flow, associated to all connectors.

Table 3: Type Labelling

element	type
Cloud	Cloud
Fog Node	Fog
IoT Field Gateway	Gateway
Intelligent Monitoring Device	

Every element and connector has multiple associated attributes. For instance attributes Authentication, Authorization and Tamper Protection are three attributes of the IoT Field Gateway element. Cloud elements have additional attributes, such as the Data Logging and Data Encryption. Data Encryption attribute can for instance take values from the domain {Yes, No, Undefined}.

---

[4] We only consider acyclic paths. This is sufficient to express all interesting threats.

B   Threat Logic

We introduce *threat logic* for specifying potential threats. While the authors of THREATGET define their own syntax and semantics to express threats [3] we use standard predicate logic in this paper. The use of the predicate logic facilitates the encoding of the forthcoming algorithms into SMT formulas. Our implementation contains an automated translation from THREATGET syntax to threat logic.

Let X be a finite set of *item* variables and P a finite set *path* variables. The syntax of threat logic is defined as follows:

$$\varphi := R(X \cup P) \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \exists p.\varphi \mid \exists x.\varphi$$

where $x \in X$, $p \in P$, and $R(X \cup P)$ is a predicate of the form:

$$
\begin{aligned}
R(X \cup P) := \; & \texttt{type}(x) = t & (1)\\
\mid \; & x_{\text{ec}} \text{ in } p & (2)\\
\mid \; & \texttt{connector}(x_e, x_c) & (3)\\
\mid \; & \texttt{src}(x_c) = x_e & (4)\\
\mid \; & \texttt{tgt}(x_c) = x_e & (5)\\
\mid \; & \texttt{src}(p) = x_e & (6)\\
\mid \; & \texttt{tgt}(p) = x_e & (7)\\
\mid \; & \texttt{crosses}(x_c, x_b) & (8)\\
\mid \; & \texttt{contained}(x_{eb}, x_b) & (9)\\
\mid \; & \texttt{holds}(x_{ec}, x_a) & (10)\\
\mid \; & \texttt{val}(x, att) = y & (11)
\end{aligned}
$$

where $x \in I$, $x_e \in E$, $x_c \in C$, $x_{ec} \in E \cup C$, $x_b \in B$, $x_{eb} \in E \cup B$, $x_a \in A$, $p \in P$, $t \in \mathbb{I}$, $att \in \mathfrak{A}$, and $v \in \mathfrak{D}(att)$.

The predicate (1) holds if the type of x is t, (2) holds if the element or the connector x_{ec} is in the path p, (3) holds if the element x_e is either the source or the target of the connector x_c, (4) holds if the source of the connector x_c is the element x_e, (5) holds if the target of the connector x_c is the element x_e, (6) holds if the source of the path p is the element x_e, (7) holds if the target of the path p is the element x_e, (8) holds if the connector x_c crosses the boundary x_b, (9) holds if the element or boundary x_{eb} is contained in the boundary x_b, (10) holds if the element or the connector x_{ec} holds the asset x_a, and (11) holds if the valuation of the attribute *att* associated to the item x is equal to y.

Example 4. Consider a requirement that states that there exists a path in the model such that all the elements in that path fulfil a certain conditions, e.g. all elements are of type Cloud. The following threat logic formula expresses the above requirement:

$$\exists p.\forall e.(e \text{ in } p \implies \texttt{type}(e) = \text{Cloud}).$$

We define an *assignment* Π_M as a partial function that assigns item and path variables to items and paths from M. We denote by $\Pi_M[x \mapsto i]$ the item

assignment in which x is mapped to i and otherwise identical to Π_M. Similarly, we denote by $\Pi_M[p \mapsto \pi]$ the path assignment in which p is mapped to π and otherwise identical to Π_M. The semantics of threat logic are defined inductively in Table 4.

Table 4: Threat logic semantics.

$$\Pi_M \models \mathsf{type}(x) = t \quad\leftrightarrow\quad \tau(\Pi_M(x)) = t$$
$$\Pi_M \models x_{ec} \ \mathsf{in} \ p \quad\leftrightarrow\quad \Pi_M(x_{ec}) \in (\mathrm{elements}(\Pi_M(p)) \cup \mathrm{connectors}(\Pi_M(p)))$$
$$\Pi_M \models \mathsf{connector}(x_e, x_c) \quad\leftrightarrow\quad \mathbf{s}(\Pi_M(x_c)) = \Pi_M(x_e) \ \mathrm{or} \ \mathbf{t}(\Pi_M(x_c)) = \Pi_M(x_e)$$
$$\Pi_M \models \mathsf{src}(x_c) = x_e \quad\leftrightarrow\quad \mathbf{s}(\Pi_M(x_c)) = \Pi_M(x_e)$$
$$\Pi_M \models \mathsf{tgt}(x_c) = x_e \quad\leftrightarrow\quad \mathbf{t}(\Pi_M(x_c)) = \Pi_M(x_e)$$
$$\Pi_M \models \mathsf{src}(p) = x_e \quad\leftrightarrow\quad e_{start}(\Pi_M(p)) = \Pi_M(x_e)$$
$$\Pi_M \models \mathsf{tgt}(p) = x_e \quad\leftrightarrow\quad e_{end}(\Pi_M(p)) = \Pi_M(x_e)$$
$$\Pi_M \models \mathsf{crosses}(x_c, x_b) \quad\leftrightarrow\quad (\Pi_M(x_b), \mathbf{s}(\Pi_M(x_c))) \notin \beta^* \Leftrightarrow (\Pi_M(x_b), \mathbf{t}(\Pi_M(x_c))) \in \beta^*$$
$$\Pi_M \models \mathsf{contained}(x_{eb}, x_b) \quad\leftrightarrow\quad (\Pi_M(x_b), \Pi_M(x_{eb})) \in \beta^*$$
$$\Pi_M \models \mathsf{holds}(x_{ec}, x_a) \quad\leftrightarrow\quad (\Pi_M(x_{ec}), \Pi_M(x_a)) \in \alpha$$
$$\Pi_M \models \mathsf{val}(x, att) = y \quad\leftrightarrow\quad v(\Pi_M(x), att) = y$$
$$\Pi_M \models \neg\varphi \quad\leftrightarrow\quad \Pi_M \not\models \varphi$$
$$\Pi_M \models \varphi_1 \vee \varphi_2 \quad\leftrightarrow\quad \Pi_M \models \varphi_1 \ \mathrm{or} \ \Pi_M \models \varphi_2$$
$$\Pi_M \models \exists x.\varphi \quad\leftrightarrow\quad \exists i \in I.\Pi_M[x \to i] \models \varphi$$
$$\Pi_M \models \exists p.\varphi \quad\leftrightarrow\quad \exists \pi \in P(M).\Pi_M[p \to \pi] \models \varphi$$

We say that a threat logic formula is *closed* when all occurrences of item variables are in the scope of a quantifier. Any closed threat logic formula is a valid threat specification. Given a system model M and a closed threat logic formula φ, we say that M *witnesses* the threat φ, denoted by $M \models \varphi$ iff $\Pi_M \models \varphi$, where Π_M is an empty assignment.

B.1 From Threat Logic To First Order Logic

We observe that we interpret threat logic formulas over system models with a finite number of elements and connectors, and hence we can eliminate path quantifiers by enumerating the elements and the connectors in the path.

F_M is the translation of the model M, where the elements, connectors, boundaries, and assets are encoded as enum sorts. Further, F_M consists of the functions τ, $\mathbf{s}$, $\mathbf{t}$ as well as the relations α and β^*. The function v is not part of F_M, because the attribute values are not hard assertions. Instead we add the following constraints to F_M to ensure attributes can have only valid values:

$$\forall x \in I, \mathfrak{a} \in \mathfrak{A}. \ (\tau(x) \notin \mathbf{a}(\mathfrak{a}) \Rightarrow v(x, \mathfrak{a}) = \bot) \wedge$$
$$\forall x \in I, \mathfrak{a} \in \mathfrak{A}. \ (\tau(x) \in \mathbf{a}(\mathfrak{a}) \Rightarrow \bigvee_{y \in \mathfrak{D}(\mathfrak{a})} v(x, \mathfrak{a}) = y)$$

The set of soft assertions $\Psi = \{F_1, \ldots, F_m\}$ is defined as

$$\{F_{i,\mathfrak{a},x}\}_{i \in I, \tau(x) \in \mathbf{a}(\mathfrak{a}), x \in \mathfrak{D}(\mathfrak{a}), x \neq v_\iota(i,\mathfrak{a})},$$

where $F_{i,\mathfrak{a},x} = \bigvee_{x' \in \mathfrak{D}(\mathfrak{a}), x \neq x'} v(i, \mathfrak{a}) = x'$.

We use v_ι to refer to the original attribute value assignment in M and v for the value assignment the SMT solver discovers. Each soft assertion is a disjunction over all possible attribute values with the exception of the possible new value. This disjunction is violated iff the attribute is indeed assigned this new value. Note that the current value of the attribute is in each disjunction and therefore keeping the current value of an attribute has cost 0. The cost of $F_{i,\mathfrak{a},x}$ is

$$cost(F_{i,\mathfrak{a},x}) = w(i, \mathfrak{a}, v_\iota(i, \mathfrak{a}), x).$$

Finally, we denote by $\hat{F}_M = F_M \wedge F_1 \wedge \ldots \wedge F_m$ the full translation of the model M that includes the soft assertions.

Definition 8. *Consider a system model M with n elements. We define the inductive translation operator T, that translates threat logic formulas to first-order formulas with their known semantics. The result of T can be fed directly into an SMT solver. Given a threat logic formula φ defined over X and P, we define a mapping σ that associates to every $p \in P$ a set $\{x_p^1, \ldots, x_p^{n-1}, k_p\}$ of fresh and unique x_p^i item (connector) variables not in X and a fresh and unique k_p integer variable. We use k_p to denote the length of the actual path discovered by the SMT solver and n for the maximum possible length of a path, where n is the number of elements in M.*

We define

$$
\begin{aligned}
T(\mathsf{src}(p) = x_e) &= x_e = \boldsymbol{s}(x_p^1) \\
T(\mathsf{tgt}(p) = x_e) &= \bigvee_{i=1}^{n-1} k_p = i \wedge x_e = \boldsymbol{t}(x_p^i) \\
T(x_{ec} \ \mathsf{in} \ p) &= \bigvee_{i=1}^{n-1} (i \le k_p \wedge (x_{ec} = x_p^i \vee \\
&\qquad x_{ec} = \boldsymbol{s}(x_p^i) \vee x_{ec} = \boldsymbol{t}(x_p^i))) \\
T(R'(X)) &= R'(X) \\
T(\neg\varphi) &= \neg T(\varphi) \\
T(\varphi_1 \vee \varphi_2) &= T(\varphi_1) \vee T(\varphi_2) \\
T(\exists x.\varphi) &= \exists x.T(\varphi) \\
T(\exists p.\varphi) &= \exists x_p^1, \ldots, x_p^{n-1} \in C, k_p \in \{1, \ldots, n-1\}. \\
&\qquad \left(\bigwedge_{i=1}^{n-1} \left(i \le k_p \Rightarrow \right. \right. \\
&\qquad (3) \ (\bigwedge_{j=1}^{i} \boldsymbol{t}(x_p^i) \neq \boldsymbol{s}(x_p^j)) \wedge \\
&\qquad \left. \left. (4) \ (i = 1 \vee \boldsymbol{t}(x_p^{i-1}) = \boldsymbol{s}(x_p^i)) \right) \right) \wedge T(\varphi)
\end{aligned}
$$

where $R'(X)$ are the other predicates defined over item variables only.

The translation of $T(\exists p.\varphi)$ ensures that $\{x_p^i\}_i$ and k_p encode a path of size k_p (see (4)) that is acyclic (i.e. no element in the encoded path is repeated, see (3)).

Proposition 1. *Let M be a system model, Π_M an assignment function and φ a closed threat logic formula. We have that*

$$\Pi_M \models \varphi \text{ iff } \hat{F}_M \wedge T(\varphi) \text{ is satisfiable}$$

Example 5. We formalize in Table 5 the two threats described informally in Section 1.

Table 5: Two threats from motivating example in threat logic.

Threat 1 $\exists e.(\mathsf{type}(e) = \text{Cloud} \wedge \mathsf{val}(e, \text{Data Logging}) = \text{Yes})$
$\wedge \mathsf{val}(e, \text{Data Encryption}) \neq \text{Yes})$
Threat 2 $\exists p, x_e^1, x_e^2.\mathsf{src}(p) = x_e^1 \wedge \mathsf{tgt}(p) = x_e^2 \wedge$
$\mathsf{type}(x_e^2) = \text{Cloud} \wedge \mathsf{type}(x_e^1) = \text{Device}$
$\wedge \mathsf{val}(x_e^2, \text{Data Logging}) \neq \text{Yes})$

B.2 Proofs

For the following proofs we recall our definition of $T(\exists p.\varphi)$.

$$T(\exists p.\varphi) = \exists x_p^1, \ldots, x_p^{n-1} \in C, k_p \in \{1, \ldots, n-1\}. \tag{1}$$

$$\left(\bigwedge_{i=1}^{n-1} \left(i \leq k_p \Rightarrow \right. \right. \tag{2}$$

$$(\bigwedge_{j=1}^{i} \mathbf{t}(x_p^i) \neq \mathbf{s}(x_p^j)) \wedge \tag{3}$$

$$\left. \left. (i = 1 \vee \mathbf{t}(x_p^{i-1}) = \mathbf{s}(x_p^i)) \right) \right) \wedge T(\varphi) \tag{4}$$

Proof. The proof works by structural induction over the threat logic formula. The base cases are if φ equals to:

- $R'(X)$: since $T(R'(X)) = R'(X)$ and this directly refers to the semantics, the property is fulfilled
- $\mathsf{src}(p) = x_e$: since $T(\mathsf{src}(p) = x_e) = (\mathbf{s}(x_1^p) = x_e)$, the property is fulfilled
- $\mathsf{tgt}(p) = x_e$: The target of a path is the target of the last connector in the path. For a known length of the path k_p the disjunction collapses to $x_e = \mathbf{t}(x_p^{k_p})$, because $k_p = i$ is false for all other i. This corresponds to the semantics of the path.
- x_{ec} in p: Using the same argument as above, all x_p^i past the length of the path k_p are ignored.

The induction cases are $\neg\varphi$, $\varphi_1 \vee \varphi_2$, $\exists x.\varphi$ and $\exists p.\varphi$. All but the last case are trivial.

To prove the correctness of the translation of $\exists p.\varphi$ we split the problem into two directions. In the first direction we proof that if there exists a path p in our model our first order logic encoding will find it. Suppose there exists a path $\pi \in P(M) = e_1, c_1, e_2, c_2 \cdots, c_{l-1}, e_l$, such that $\Pi_M[p \rightarrow \pi] \models \varphi$. We show that then $F_M \wedge T(\exists p.\varphi)$ is satisfiable by giving values for the variables $x_p^1, \ldots, x_p^{n-1}, k_p$:

$$
\begin{aligned}
k_p &= l - 1 \\
x_p^1 &= c_1 \\
&\vdots \\
x_p^{l-1} &= c_{l-1} \\
x_p^l &= c_1 \\
&\vdots \\
x_p^{n-1} &= c_1
\end{aligned}
$$

The condition $i \leq k_p$ ensures that all x_p^i for $i > l - 1$ are ignored in the following checks. For $1 \leq i < j \leq l - 1$ condition (3) is clearly fulfilled as π is an acyclic path. Condition (4) follows because $c_1, \ldots, c_{l-1}$ form a path. $T(\varphi)$ follows by induction.

For the second direction we assume an assignment to $x_p^1, \ldots, x_p^{n-1}, k_p$ fulfilling conditions (1-4) and we need to show that there exists a path $\pi \in P(M)$, such that $\Pi_M[p \rightarrow \pi] \models \varphi$. We construct the path $\pi = \mathbf{s}(x_p^1)x_p^1\mathbf{t}(x_p^1)x_p^2\mathbf{t}(x_p^2)\ldots x_p^{k_p}\mathbf{t}(x_p^{k_p})$. This is a valid path because $\mathbf{s}(x_p^i) = \mathbf{t}(x_p^{i-1})$ for all $1 < i \leq k_p$ due to condition (4). It is also an acyclic path due to condition (3). φ follows by induction. $\qquad\square$

Theorem 1. *Let M be a system model and $\{\varphi_1, \ldots, \varphi_n\}$ a set of closed threat logic formula. We have that $\max \mathrm{solve}(F_M \wedge \bigwedge_{i=1}^n \neg\varphi_i \wedge \bigwedge_{F \in \Psi} F)$ provides the solution to the minimum attribute repair problem.*

Proof. Given a system model M, its encoding $F_M \wedge \bigwedge_{F \in \Psi} F$ is satisfiable by definition. Suppose a set of threat rules $\{\varphi_1, \ldots, \varphi_n\}$, we have three cases:

1. If $M \models \bigwedge_{i=1}^n \neg\varphi_i$ then $F_M \wedge \bigwedge_{F \in \Psi} F \wedge \bigwedge_{i=1}^n \neg T(\varphi_i)$ is satisfiable.
2. If $M \not\models \bigwedge_{i=1}^n \neg\varphi_i$ and $\mathrm{solve}(F_M \wedge \bigwedge_{i=1}^n T(\neg\varphi_i)) = \mathrm{unsat}$, then there exists φ_i that cannot be repaired and reveals a structural threat regardless of the attributes.
3. If $M \not\models \bigwedge_{i=1}^n \neg\varphi_i$ and $\mathrm{solve}(F_M \wedge \bigwedge_{i=1}^n T(\neg\varphi_i)) = \mathrm{sat}$ then $\exists \Psi' \subseteq \Psi$ s.t. $\mathrm{solve}(F_M \wedge \bigwedge_{i=1}^n T(\neg\varphi_i) \wedge \bigwedge_{F \in \Psi'} F) = \mathrm{unsat}$, and $\max \mathrm{solve}(F_M \wedge \bigwedge_{i=1}^n T(\neg\varphi_i) \wedge \bigwedge_{F \in \Psi'} F) = \mathrm{sat}$ with some cost k greater than zero.

Consider case (3). Suppose that the minimum attribute threat repair for M (with valuation function v) and $\{\varphi_1, \ldots, \varphi_n\}$ has cost $k' < k$. By definition of minimum attribute threat repair, there exists a valuation function v', such that

$M[v'\backslash v] \models \wedge_{i=1}^{n} \neg\varphi_i$ and $d(v, v') = k'$. Let $\Psi' = \{F_{i,\mathfrak{a},x} \mid v(i, \mathfrak{a}) \neq v'(i, \mathfrak{a})\}$ be the set of soft assertions for the item-attribute pairs $(i, \mathfrak{a})$ for which the valuation changes from x to x' with $cost(F_{i,\mathfrak{a},x'}) = w(i, \mathfrak{a}, x, x')$, i.e. $\Sigma_{F\in\Psi'} cost(F) = k'$. It follows that $\text{solve}(F_M \wedge \bigwedge_{i=1}^{n} T(\neg\varphi_i) \wedge \bigwedge_{F\in\Psi\backslash\Psi'} F) = \text{sat}$, hence $\max\text{solve}(F_M \wedge \bigwedge_{i=1}^{n} T(\neg\varphi_i) \wedge \bigwedge_{F\in\Psi'} F) = \text{sat}$ with cost k', which is a contradiction. It follows that MaxSAT provides the solutions to the minimum attribute repair problem.

$\square$

C Case Study: Key Fob

This case study consists of a simplified architectural model of a remote locking and unlocking mechanism in a car, depicted in Figure 2. This model illustrates the main capabilities of the THREATGET tool. It consists of a key fob that communicates in a wireless fashion with the car's lock/unlock system.
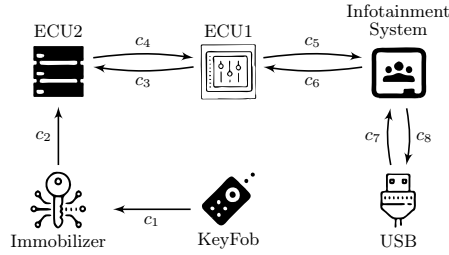

Fig. 2: Example modeled in THREATGET

This system is connected to an ECU (Electronic Control Unit) (ECU2 in Figure 2) that implements the actual locking and unlocking functionality. After processing the signal, the user gets feedback (e.g., a flashing light). The ECU realizing the lock/unlock feature is connected (via a CAN bus) to an ECU that controls the Infotainment System. The Infotainment System interacts with the driver via a USB port. Components (elements) and connectors in the model have attributes associated to them, and these attributes have values. For instance, the connector between the Key Fob and the Lock/Unlock system has an Encryption attribute set to *false*, meaning that no encryption is used to communicate between these two elements. Threats are structural security weaknesses in the model that are modeled as relations between entities in the model (elements, connectors, attributes and their values, etc.). A potential threat in the remote locking model could be the presence of unprotected (non-encrypted) wireless communication channels. The model has a total of 6 elements and 8 connectors, without any assets or security boundaries.

The threat repair procedure results in an optimized configuration of the system architecture model with security attributes that prevent threats from the original model. The implementation of the recommended measures has an immediate positive impact on the system security. We illustrate one of the proposed prevention measures.

Manipulation of vehicular data: this threat is present because a malicious user could connect a Linux machine to the USB interface, access the CAN bus via the Infotainment system, manipulate command data and send them to the ECUs

Table 6: Results of attribute repair applied to the Key Fob case study.

	All threats		Subset of threats	
	full	h-partial	full	h-partial
verdict	UNSAT	SAT	SAT	SAT
total # formulas	165	165	21	21
# repairable formulas	n/a	25	4	4
# unrepairable formulas	n/a	7	0	0
# formulas w/t threat	n/a	133	17	17
total cost	n/a	33	9	11
time (s)	3.65	103	9.58	25.65

through the CAN bus. The recommended prevention measure consists in implementing authentication for connections to the USB interface, which would result in ignoring unauthenticated command messages.

We summarize the evaluation of this case in Table 6. We first tried the vanilla procedure (Algorithm 1). The procedure gave an UNSAT verdict in 3.65s, despite the fact that all unrepairable rules (according to the has_attr() syntactic check) were removed from the optimization. This result indicates that that there are a-priori repairable threat formulas that are either inconsistent with the system model or with another threat formula. We note that the UNSAT outcome is not very useful to the engineer because it does not give any (even partial) repair. On the other hand, the heuristic method (Algorithm 2) successfully computed a partial repair of the system model, repairing 25 out 32 threats in 103s.

In the next step, we manually selected 21 threat formulas that could be simultaneously repaired by the vanilla method. The vanilla procedure found 4 threats and repaired all of them in 9.58s with the cost on 9. The heuristic method also found 4 threats, repaired all of them, but with the sub-optimal cost of 11 in 25.65s. We can see that the flexibility of the heuristic method comes at a price - both in terms of optimality of the solution and computation time.
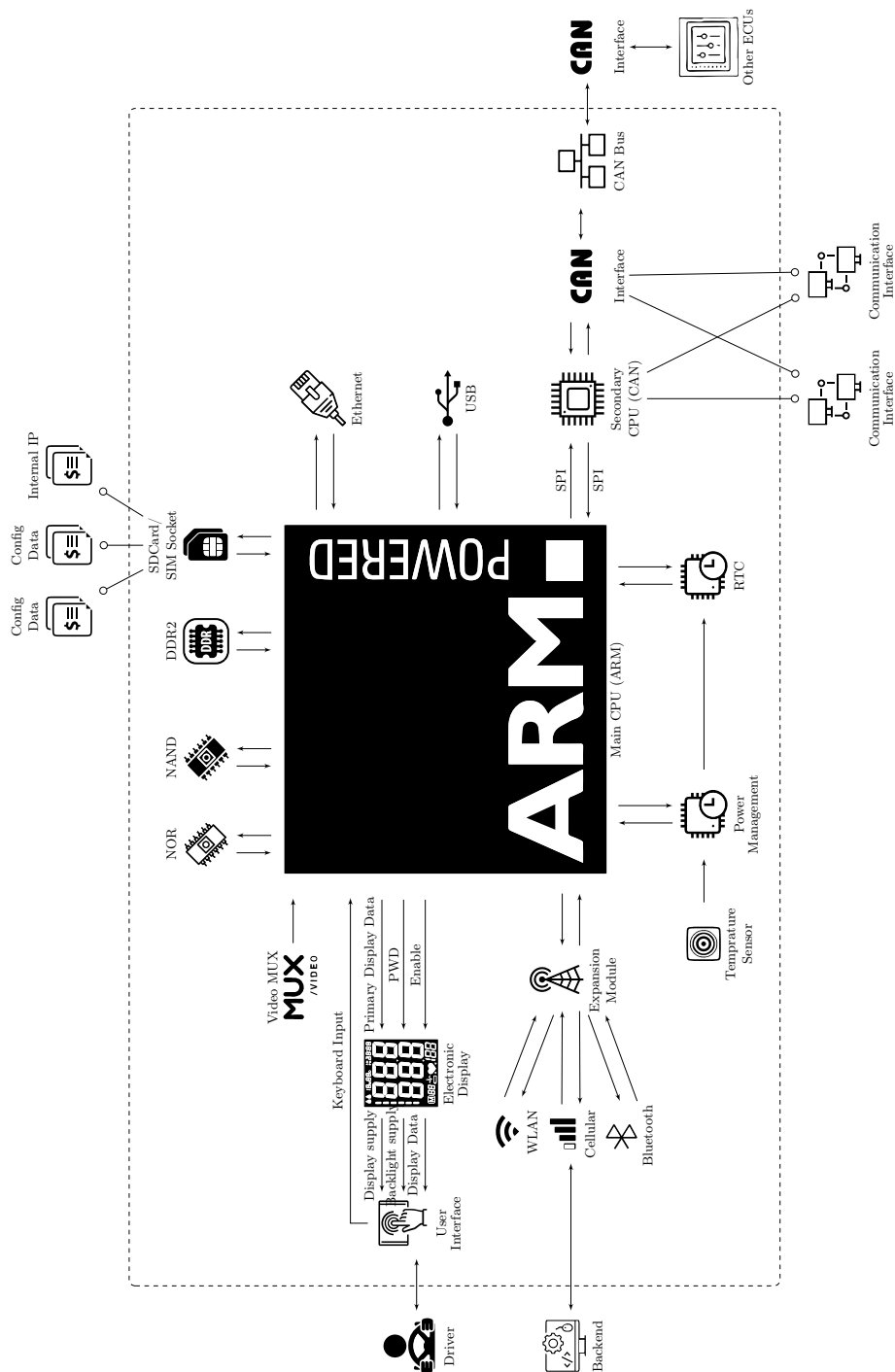
D Case Study: Vehicular Telematic Gateway

Fig. 3: Model of Vehicular Telematic Gateway