On Threat Model Repair

Roderick Bloem[1], Sebastian Chlup[2], Dejan Ničković[2(✉)], and Christoph Schmittner[2]

[1] Graz University of Technology, Graz, Austria
[2] AIT Austrian Institute of Technology,Seibersdorf, Austria
dejan.nickovic@ait.ac.at

Abstract. Security by construction is an approach to system development where security considerations are integrated into the design process from the very beginning. Threat modeling helps identify potential threats and vulnerabilities early in the system development process, assess the risk associated with each threat, and design appropriate mitigation actions. In this paper, we study threat model repair, a method to automatically suggest structural changes to the design that mitigate threats discovered by the analysis. This helps find a secure design early in the process by allowing a user to quickly iterate over different design variants.

1 Introduction

With the advent of the Internet of Things (IoT), communication-based technologies have penetrated many new industrial domains. The Vehicle-to-X (V2X) paradigm in the automotive sector, intelligent energy distribution systems in the smart grids, and the distribution of manufacturing supply chains are just a few examples of modern applications where communication plays a key role. This tremendous increase in connectivity comes at a price of expanded cybersecurity threats. Hence, there is an imminent need to take security-related decisions into account when developing new applications from the earliest stages of design. This urgency has been widely recognized by the researchers and the practitioners. It has led to initiatives such as the upcoming ISO/SAE 21434 standard that calls for a more principled security assessment of designs.

Security by construction is a methodology that brings security considerations to the forefront of the system development process. There are multiple complementary facets to this holistic approach to security. In the early stages of the system design, security requirements are identified and documented and the system architecture is modeled to minimize attack surfaces. In parallel, threat modeling is used to identify potential threats and their associated risks. System development is accompanied by the assessments of the code, using static and dynamic verification and analysis tools to detect potential security issues. Security testing, which includes fuzz testing and penetration testing, becomes an integral part of the continuous integration and deployment (CI/CD) pipeline.

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2025
T. Margaria and B. Steffen (Eds.): ISoLA 2024, LNCS 15222, pp. 302–310, 2025.
https://doi.org/10.1007/978-3-031-75387-9_18

Threat modeling and analysis are important pieces of the security by construction methodology, especially in the earliest phases of the development process. Threat modeling consists in collecting and formalizing rules that describe potential threats and vulnerabilities originating from documented threats, standards, and domain expert knowledge. We can use threat analysis to detect security issues in the system architecture with respect to the modeled threats. Today, there is a landscape of methods and tools that provide threat modeling and analysis capabilities. Microsoft Threat Modelling Tool (MTMT) [McR14] has been developed as a visual system structure modeling tool as part of their Security Development Lifecycle. THREATGET [SSK19, CT21] is another threat modeling, analysis and risk management tool that has been mainly used in the automotive domain.

A designer can use vulnerabilities that were identified with threat analysis to update the system architecture in a way that addresses and mitigates the found threats. We have introduced *threat model repair* as a technique to automatically suggest possible changes to the system model to resolve and remove detected threats. The designer can take these suggestions to update the actual design according to their preferences. In a previous paper [TEK+23], we formalized the problem for the case that the repair is limited to the security attributes associated to the system architecture components and communication links. A possible suggestion would then be to change a communication link from unencrypted to encrypted.

Changes to attributes values severely limit the type of repairs that we can suggest, thus missing useful ways to change the design. For instance, suppose we have a security domain protected by a firewall. Suppose that there is a security threat that consists of many components inside the security domain all communicating to an outside component X over an unencrypted connection. A possible solution to this problem, which can be realized by changing attributes, is to redesign all communication links to be encrypted. However, it may be much cheaper to move component X into the protected domain, obviating the need for encryption hardware at all components. This, however, is a structural change that is not in the scope of our previous method.

In this paper, we present a procedure for threat model repair that uses optimization modulo theories to propose changes in the system architecture that remove potential threats. We consider more general system repairs that include the addition of components or restructuring of the system architecture. We also discuss how to avoid trivial changes to the architecture that achieve security by sacrificing functionality.

2 System and Threat Models

Threat analysis and repair is typically performed on a *system model*. System architectures can be modeled at different levels of abstraction and we adopt the model used by the THREATGET tool. We will recapitulate the formalization and refer to [TEK+23] for details. A system model S consists of:

- a set E of *elements*: an element $e \in E$ is a typed logical or physical component. Software and databases are two examples of logical components, while sensors, actuators, ECUs and firewalls are physical components.
- a set $C \subseteq E \times E$ of *connectors*: a connector $c \in C$ is communication link between a *source* and a *target* elements that has a *type* such as *wired* or *wireless*. We refer to elements and connectors jointly as *components*.
- a set A of *security assets* with an associated relation holds: an asset $a \in A$ describes components that need to be protected from malicious access. Every element and connector hold multiple assets. Similarly, each asset can be held by multiple elements and connectors.
- a set $B \subseteq 2^E$ of *security boundaries*: a boundary $b \in B$ describes a separation between logically, physically, or legally separated system elements.
- a set Δ of *attributes*. Every attribute has an associated domain and a relation v that associates an attribute value to a component. An attribute $\delta \in \Delta$ is a property that is associated to system elements, connectors and/or assets. Attributes assume a value from their associated domain. We denote by $v(x, \delta)$ the value of the attribute δ associated to the component x. Finally, we assign a *cost* of changing $v(x, \delta)$ to another attribute value $v'(x, \delta)$. The cost of leaving a value unchanged is zero; all pther costs are strictly positive.

Example 1. We use a simplified remote locking and unlocking mechanism in a car, depicted in Fig. 1, to illustrate the system model. It consists of a key fob that communicates in a wireless fashion with the car's lock/unlock system.

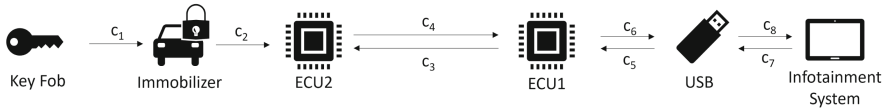

Fig. 1. Example modeled in THREATGET

This system is connected to an ECU (Electronic Control Unit) (ECU2 in Fig. 1) 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. The model has a total of 6 elements and 8 connectors, without any assets or security boundaries.

Threats are modelled as *rules*, where we use t to denote a threat rule and we group multiple threat rules into a set T, which we call the threat *database*. We

adopt *threat logic*, a simplified variant of a predicate logic, inspired by THREAT-GET's own language to express threats. The predicates in the threat logic are defined over variables and constants that refer to the different components of the system model.

Example 2. Consider a threat rule that describes a vulnerability consisting of a path in the system model from a USB interface to a cloud such that the cloud does not use encryption. This rule is formalized as follows. Note the quantification over elements and connectors, the use of source and target to describe the source and the target of a connector, the use of type for the type of the element, and the use of encryption to describe the encryption attribute of the element e_2 with value None:

$$\exists e_1, e_2, c.(\mathsf{source}(c) = e_1 \ \wedge \ \mathsf{target}(c) = e_2 \ \wedge \ \mathsf{type}(e_2) = \text{USB\_Interface} \wedge \mathsf{type}(e_2) = \text{Cloud} \ \wedge \ \mathsf{encryption}(e_2) = \text{None}.$$

3 Threat Model Analysis and Repair

3.1 Threat Analysis

Threat analysis is a process for identifying components and other assets in a system that are prone to security risks and hence need to be protected in terms of confidentiality, integrity, availability, and other security-related properties. In the context of this paper, we formalize threat analysis as follows.

Threat Analysis

Given a system model S and a threat database T, find all the threat rules $T' \subseteq T$ to which S is vulnerable.

With the formal representations of the system architecture model and the threat rules, threat analysis can be naturally stated as a satisfiability (SAT) problem, i.e., the problem of determining if a propositional formula can be evaluated to true by some assignment of truth values of its variables. We encode the entire system model with a Boolean formula φ_S and each threat rule $t \in T$ with a formula φ_t that formulates the presence of the threat t. Checking the presence or the absence of the threat t in the system model S then consists of applying the SAT solver to the closed formula $\varphi_S \wedge \neg\varphi_t$. (Note the negation of the treat formulas to denote absence of a threat). The *sat* (*unsat*) verdict indicates the absence (presence, resp.) of the threat t in the system model S and consequently, the set T' of threats to which S is vulnerable is given as $\{t \in T \mid \mathsf{solve}(\varphi_S \wedge \varphi_t) = \text{unsat}\}$. If the system is correct, the formula $varphi_{\hat{S}} \wedge \bigwedge_{t \in T} \neg\varphi_t$ is satisfied.

3.2 Threat Repair

Threat repair is a mechanism that automatically alters a system model in a way that ensures the absence of threats, and hence has more ambitious goals

compared to the (passive) threat analysis process. There are many ways that a system model can be adapted to fulfil a certain goal, and we are typically interested in a repair that is *optimal* with respect to some notion of cost. We formalize the optimal threat repair as follows.

Optimal Threat Repair

Given a system model S and a threat database T such that there is at least one threat rule $t \in T$ satisfied by S, find another system model S' derived from S such that no rule in T satisfies S' and that the change from S to S' incurs a minimal cost.

We distinguish between two flavors of threat repair – security attribute repair and full model repair. A security attribute repair consists of changing the value of one or more security attributes of the system model, without altering its structure. A full model repair may also include new components, remove existing components, or change components such as security boundaries.

The repair problem is more ambitious than its analysis counterpart and in this case SAT solving is not sufficient. Instead we use *maximum satisfiability* (MaxSAT) to formulate our repair problem. MaxSAT generalizes SAT by solving the problem of finding an assignment to the variables of the formula that satisfies the maximum number of clauses. In the *weighted* variant of MaxSAT, a cost is associated to each clause and the goal to minimize the total sum of individual costs rather then to just count the number of satisfied clauses. Weighted MaxSAT is formalized with the following definition.

Definition 1 (Weighted MaxSMT [BP14]**).** 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 $K \subseteq \{1, \ldots, m\}$ such that: (1) $\mathbf{F} \wedge \bigwedge_{k \in K} F_k$ is satisfiable, and (2) the total cost $\sum_{k \in K} cost_k$ is minimized.

(Note the assumption that the solver minimizes, which makes the formulation below simpler.)

In the case of the security attribute repair, we partition our system architecture model formula φ_S into a set of formulas $\{\varphi_{\hat{S}}, \psi_1, \ldots, \psi_m\}$, where $\varphi_{\hat{S}}$ describes the structure of the system architecture model and every ψ_i encodes a security attribute assignment in the model. We associate costs to each ψ_i according to the change it suggests: If ψ_i encodes an assignment of value v' to attribute a of component c and its original value is v, it carries the cost of changing a from v to v' (which is 0 if $v = v'$). Following the notation from Definition 1, we have that $\mathbf{F} = \varphi_{\hat{S}} \wedge \bigwedge_{t \in T} \neg\varphi_t$ and $F_i = \psi_i$. We call a clause in $\mathbf{F}$ or a formula $\neg\varphi_t$ a *hard assertion* and the F_i are *soft assertions*. Intuitively, the solution $\mathsf{max\_solve}(\varphi_{\hat{S}} \wedge \bigwedge_{t \in T} \neg\varphi_t, \{\psi_1, \ldots, \psi_m\})$ gives one of three verdicts:

– sat verdict with total cost 0: this means that the system model has no threats and no repair is required (recall that repairs have positive costs),

- sat verdict with total cost k: this means that the system model has threats that can be removed by changing security attributes. The total cost of the required changes is k. In addition, the solver gives a suggestion for the new security attribute values that are required to remove the threats,
- unsat verdict: the system model cannot be repaired by changing security attributes and structural changes are required to remove the threats.

3.3 Full System Repair

While the security attributes of the system components cover a large scope of security-relevant properties, there are threats that require more structural adaptations of the system architecture. Therefore, there is a need to complement security attributes repair with more general model repair approaches. However, the ability of arbitrary addition and removal of system components, communication links and security boundaries may render the repair process too powerful, resulting in uninteresting repairs. For instance, removing all communication links between components can be a system repair that effectively removes the vast majority of potential threats, but results in a worthless system without functionality.

Meaningless repair strategies can be minimized by restricting the space of possible repairs to a set of templates that are commonly used by security engineers. In this paper, we have conducted a preliminary study to identify a library of useful repair templates in the context of the system and the threat model used. We identify three key structural repairs that each aim to separate trusted from untrusted components in different ways. Together, these repairs account for many of the structural repairs we see in practice.

Firewalling. The first structural repair consists in separating the trusted (safety-critical) part of the system from the untrusted one (e.g. entertainment system) with a firewall or a secure gateway. For example, safety-critical elements in a car that are connected to a Controller Area Network (CAN) bus cannot be done just by setting appropriate security attributes. The real protection of the safety-critical components would require their separation from the untrusted part of the system.

Introducing a security boundary. Another structural repair consists in separating the trusted and safety-critical part of the system from the untrusted one by using a security boundary. We note that a security boundary is a more abstract concept that is used during the design phase of the system, and that its realization is typically done using concrete mechanisms protecting the interface between the trusted and the untrusted parts of the system.

Network segregation. Segregation is another structural repair strategy that allows to effectively protect system assets, improve network efficiency, and meet compliance requirements. For example, a car system in which both safety-critical components such as braking and steering are connected to the same bus as non-critical components such as lighting, infotainment, or windows. In this case, we can introduce multiple CAN buses to segregate the

network according to the criticality of a component. We can use a secure gateway to connect and manage communication between these two parts of the system.

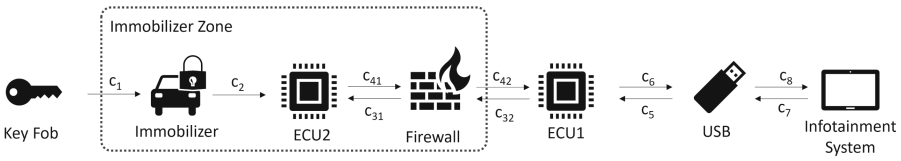

Fig. 2. Structural repair of the Key Fob model.

Example 3. The key fob model from Fig. 1 contains security weaknesses that can be mitigated by changing security attributes as well as weaknesses that require a structural change. As an example of the first type, consider unprotected (non-encrypted) wireless communication channels. We can mitigate this threat by setting an appropriate security attribute to the wireless connectors.

As an example of a problem that cannot be solved using attribute changes alone, consider the infotainment system, which allows outside (potentially malicious) users to access ECU1 through the USB interface. The non-critical ECU1, in turn, is directly connected to ECU2, which implements the safety-critical lock/unlock functionality, This presents a security risk that is identified by THREATGET. This issue can be addressed by (1) adding a firewall between the two ECUs, or (2) isolating all safety-critical components (including the firewall) within a secure boundary, as shown in Fig. 2.

We adapt the weighted MaxSAT formulation of the threat repair strategy from Sect. 3.2 to allow full system repair. Suppose that a structural aspect of the system is described by a formula ϕ_{ori}. We implement the formulation of the repair in two steps.

1. We augment the ϕ_{ori} by one or more formulas $\phi_{\text{alt},1}, \ldots, \phi_{\text{alt},n}$ that describe alternative structural implementations.
2. We encode n mutually exclusive soft constraints ψ_1 through ψ_n, where ψ_i describes that repair i is selected. Each soft constraint carries the cost that is associated with the structural change.

We illustrate this idea on the example of adding a firewall between a trusted and an untrusted ECU. Consider the following threat rule t, which states that there is a threat if an untrusted ECU is directly connected to a trusted ECU and both are inside the same security boundaries:

$$\begin{aligned}\exists e_1, e_2, c.\ &\mathsf{source}(c) = e_1 \wedge \mathsf{target}(c) = e_2 \wedge \\ &\mathsf{type}(e_1) = \mathsf{ECU} \wedge \mathsf{type}(e_2) = \mathsf{ECU} \wedge \\ &\mathsf{trusted}(e_1) \wedge \neg\mathsf{trusted}(e_2) \wedge \forall b \in B : e_1 \in B \leftrightarrow e_2 \in B.\end{aligned}$$

Consider the formula φ_{ori} that encodes the connection c_3 from Fig. 1, and the formula φ_{alt} that encodes the associated firewall repair:

$$\begin{aligned}\varphi_{\text{ori}} = {} & \mathsf{source}(c_3) = \mathsf{ECU1} \wedge \mathsf{target}(c_3) = \mathsf{ECU2} \wedge \\ & \mathsf{type}(\mathsf{ECU1}) = \mathsf{ECU} \wedge \mathsf{type}(\mathsf{ECU2}) = \mathsf{ECU} \wedge \\ & \neg\mathsf{trusted}(\mathsf{ECU1}) \wedge \mathsf{trusted}(\mathsf{ECU2}), \text{ and} \\ \varphi_{\text{alt}} = {} & \mathsf{source}(c_{31}) = \mathsf{ECU1} \wedge \mathsf{target}(c) = \mathsf{Firewall} \wedge \\ & \mathsf{source}(c_{32}) = \mathsf{Firewall} \wedge \mathsf{target}(c_{32}) = \mathsf{ECU2} \wedge \\ & \mathsf{type}(\mathsf{ECU1}) = \mathsf{ECU} \wedge \mathsf{type}(\mathsf{ECU2}) = \mathsf{ECU} \ \wedge \mathsf{type}(\mathsf{Firewall}) = \mathsf{Firewall} \\ & \neg\mathsf{trusted}(\mathsf{ECU1}) \wedge \mathsf{trusted}(\mathsf{ECU2}).\end{aligned}$$

We can now adapt the system model by replacing the hard assertion φ_{ori}, with two soft assertions $\psi_1 = \varphi_{\text{ori}} \wedge \neg\varphi_{\text{alt}}$ (no repair formula) and $\psi_2 = \neg\varphi_{\text{ori}} \wedge \varphi_{\text{alt}}$ (repaired formula), with associated costs $c_1 = 0$ and $c_2 = k$. We note that the non-repaired and the repaired formulas are mutually exclusive and that by associating a positive cost to ψ_2, we will use it only in the presence of threat. The other repair templates can be encoded as weighted MaxSAT problems in a similar fashion.

We can encode a third option of introducing a security boundary in much the same way, with a different associated cost. The MaxSMT solver will then select the less costly option. Note that the possible repairs can be listed in a relatively general manner. For instance, we can introduce a new security boundary B without describing which elements are inside the region and which are outside. A security boundary can be introduced by adding a variable for each element that encodes whether the element is inside or outside the boundary. If the SMT solver decides to use the boundary to repair the threat, it can only do so by placing the trusted ECU on one side of the boundary and the untrusted ECU on the other. In all other cases, the threat will remain. The other elements will be placed by the solver in some way that respects any further threat formulas. Thus, it is not necessary to enumerate the possible placements of the components with respect to the boundary.

4 Conclusions

The problem of threat model repair can be effectively solved with the combination of traditional artificial intelligence (AI) and optimization techniques, namely SAT and weighted MaxSAT. There are however limitations to the general repair of system models. First, we showed that there are restrictions on the meaningful structural changes. Second, threat model repair is applied during (early) concept design phase. Hence, any repair at this level of abstraction must be followed and implemented in the real system. Consequently, our threat model repair approach shall be used as an engineering decision-making support and not a push-button solution.

Acknowledgements. This work received funding from the AIMS5.0 project regarding AI based transformation of the European Industry. The AIMS5.0 project is supported by the Chips Joint Undertaking and its members, including the top-up funding

by National Funding Authorities from involved countries under grant agreement no. 101112089.

References

[BP14] Bjørner, N.S., Phan, A.D.: νz - maximal satisfaction with Z3. In: Temur Kutsia and Andrei Voronkov, editors, 6th International Symposium on Symbolic Computation in Software Science, SCSS 2014, Gammarth, La Marsa, Tunisia, December 7-8, 2014, vol. 30 of EPiC Series in Computing, pp 1–9. EasyChair (2014)

[CT21] Christl, K., Tarrach, T.: The analysis approach of threatget. *CoRR*, abs/2107.09986 (2021)

[McR14] McRee, R.: Microsoft threat modeling tool 2014: identify and mitigate. Inf. Syst. Secur. Assoc. J. 39–42 (2014)

[SSK19] El Sadany, M., Schmittner, C., Kastner, W.: Assuring compliance with protection profiles with threatget. In: Alexander B. Romanovsky, Elena Troubitsyna, Ilir Gashi, Erwin Schoitsch, and Friedemann Bitsch, editors, Computer Safety, Reliability, and Security - SAFECOMP 2019 Workshops, ASSURE, DECSoS, SASSUR, STRIVE, and WAISE, Turku, Finland, September 10, 2019, Proceedings, volume 11699 of Lecture Notes in Computer Science, pp. 62–73. Springer (2019). https://doi.org/10.1007/978-3-030-26250-1_5

[TEK+23] Tarrach, T., Ebrahimi, M., König, S., Schmittner, C., Bloem, R., Nickovic, D.: Attribute repair for threat prevention. In: Jérémie Guiochet, Stefano Tonetta, and Friedemann Bitsch, editors, Computer Safety, Reliability, and Security - 42nd International Conference, SAFECOMP 2023, Toulouse, France, September 20-22, 2023, Proceedings, vol. 14181 of Lecture Notes in Computer Science, pp. 135–148. Springer (2023). https://doi.org/10.1007/978-3-031-40923-3_11