Towards security threats that matter

Katja Tuma[1], Riccardo Scandariato[1], Mathias Widman[2], and Christian Sandberg[3]

[1] University of Gothenburg, Sweden
[2] Wireless Car, Gothenburg, Sweden
[3] Volvo AB, Gothenburg, Sweden

Abstract. Architectural threat analysis is a pillar of security by design and is routinely performed in companies. STRIDE is a well-known technique that is predominantly used to this aim. This technique aims towards maximizing completeness of discovered threats and leads to discovering a large number of threats. Many of them are eventually ranked with the lowest importance during the prioritization process, which takes place *after* the threat elicitation. While low-priority threats are often ignored later on, the analyst has spent significant time in eliciting them, which is highly inefficient. Experience in large companies shows that there is a shortage of security experts, which have limited time when analyzing architectural designs. Therefore, there is a need for a more efficient use of the allocated resources. This paper attempts to mitigate the problem by introducing a novel approach consisting of a risk-first, end-to-end asset analysis. Our approach enriches the architectural model used during the threat analysis, with a particular focus on representing security assumptions and constraints about the solution space. This richer set of information is leveraged during the architectural threat analysis in order to apply the necessary abstractions, which result in a lower number of significant threats. We illustrate our approach by applying it on an architecture originating from the automotive industry.

Keywords: Architectural threat analysis, Security assets, STRIDE

1 Introduction

In an ever more complex Cyber-Physical System (CPS) domain, security and trust management are becoming burdensome for many organizations. New software products and frameworks are intended to support functionalities that handle privacy and security of sensitive data. Furthermore, the longevity of a CPS product is typically high (e.g., in the automotive, 25 years), which makes building a secure solution a substantial challenge. Security by design requires addressing security-related issues throughout the entire software development life-cycle. This paper focuses on the early stage of conceptualization of a software system, i.e., the architectural design. Planning for security in early design phases helps designers to steer the product development in a direction where threat mitigations are possible.

In particular, threat analysis is a method that strives towards validating the software architecture and discovering potential design weaknesses. This validation technique is an essential pillar of software security, together with other code-level verification techniques like static analysis and security testing [17]. For instance, Microsoft's STRIDE is a well-known and used technique to perform architectural threat analysis [20]. STRIDE and similar techniques (like LINDDUN [25]) follow the so-called software-centric approach. Such techniques center the analysis around a model of the system that resembles a graph and represents the software components (both computation and storage nodes) and the information exchanged between them (edges). From a syntactical perspective, Data Flow Diagrams (or DFD) are often used to represent such models. According to STRIDE the analysis proceeds by exploring the diagram and discovering several potential threats at each location. On one hand, this way of performing a security assessment has the benefit of being systematic. On the other hand, the analysis is prone to being repetitive and very time consuming. Empirical evidence shows that with only a handful of software components, the analysis can result in the discovery of 50 to 60 security threats, which means a scale factor of 10 [18]. This is known as the 'threat explosion' problem.

The experience of our industrial partners is that, trading systematicity for a timely discovery of most important threats is advantageous. As resources are scarce and time is limited, systematicity is considered an obstacle if it leads to 'wasting' time with security threats that are deemed as not important later on. We also learned that in the early design phase, stakeholders reason about security with a close eye on system assets. Rather than focusing the analysis on the software assets (e.g., software components and data stores), analysts observe information assets and how they move through the system. They do that by analyzing end-to-end usage scenarios which involve a certain information asset and trace the software components that are involved with exchanging, using and storing that particular asset. At each encountered software component, they reason about the potential threats to the asset.

In this paper we synthesize the lessons learned while analyzing the threats in an architecture of a connected vehicle and suggest a novel way of approaching threat analysis. We propose an architectural view for security that is based on DFDs, extended with end-to-end flows representing the information assets in the system. In our enriched model, assets are also annotated with their importance and with security objectives associated with them (e.g., confidentiality). The extended model also includes additional information that is routinely used during the threat analysis process, namely, security assumptions. The extended notation is used to guide the threat analysis and reduce the amount of 'uninteresting' threats that are found. For instance, if an end-to-end flow refers to an information asset that needs to stay confidential, it would be better to focus on disclosure threats (which directly impact confidentiality) rather than on denial-of-service threats (which impact availability instead).

In order to illustrate our approach, the first author applied it on the architecture provided in the context of the HoliSec project [4] on security of connected

vehicles. The results of the analysis have been submitted to a domain expert with extensive security background (the third author). Our discussions concluded that our approach indeed led to the identification of valid threats, likely to have the most impact to the organization in reality. Clearly, the obtained results serve as a proof of concept and are a stepping stone for future work.

The remainder of this paper is structured as follows. Section 2 describes the running example. Section 3 presents the extended notation and the needs of the analyst, Section 4 presents the guidelines for abstracting the architectural model (DFD) applied to the running example and Section 5 identifies the related work. Finally, Section 6 includes a discussion and future work, followed by concluding remarks, presented in Section 7.

2 Running example

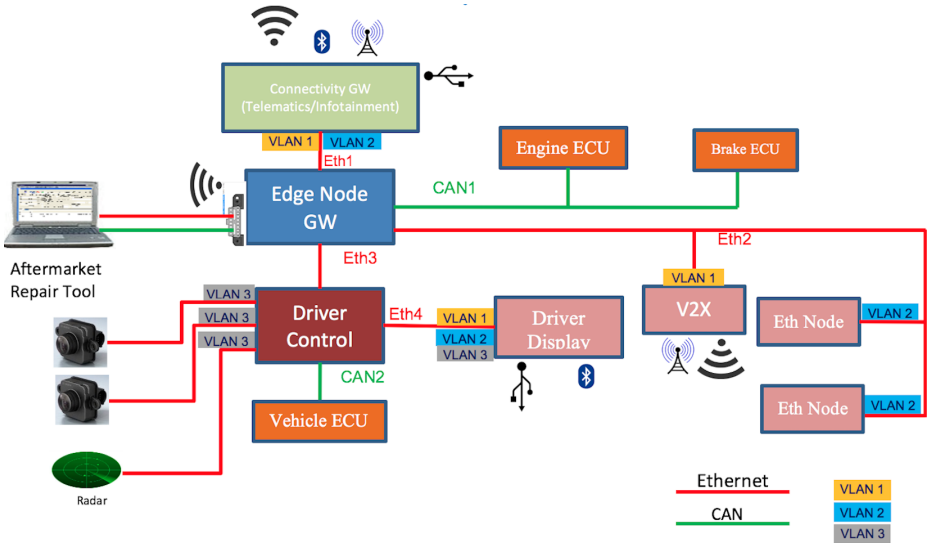
In this section we describe the architecture of a vehicle as shown in Figure 1. This example is used throughout the paper to illustrate the benefits of our proposed approach. Due to non-disclosure concerns, the example is realistic but does not correspond to an architecture of an existing product on the market.

Modern vehicles are highly complex systems, comprised of hundreds of different components called Electronic Control Units (ECUs), which are responsible for one or more particular features of the vehicle. Individual ECUs are connected to a Controller Area Network (CAN) bus, which is currently the most used in-vehicle communication protocol and also a very common target of attack [26].

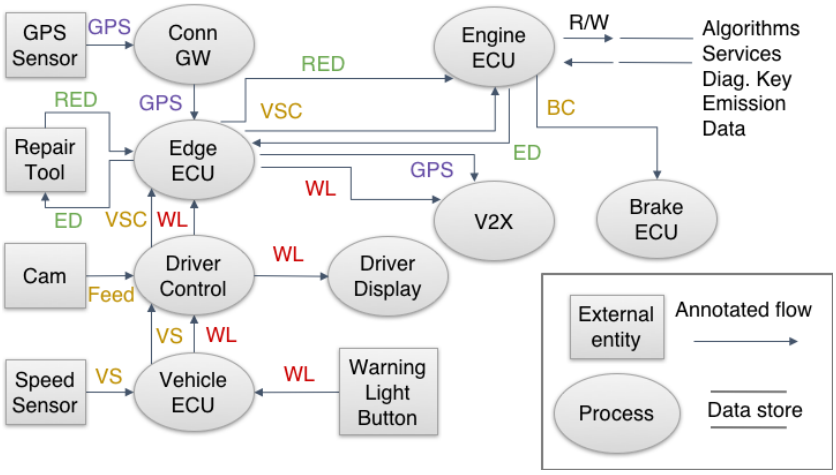
As depicted in Figure 1a, the architecture is composed of several ECUs, sensors and actuators exchanging data between each other following a specific communication protocol. The communication between individual components is further specified with a communication matrix (not shown here) of signals, source and receiver components, networks used and type of communication (e.g., broadcast or unicast). For instance, the warning light signals are broadcast on networks CAN 2, Eth 2, Eth 3 and Eth 4. The architecture in Figure 1a supports a number of functional scenarios, which are described below. Figure 1b presents a DFD, derived from the architectural information and the assets described in the scenarios. Notice that functional elements represent *processes*, while *data stores* represent the places where information is stored for later retrieval. Everything that is outside of the system (e.g., 3rd party systems) is modeled by means of *external entities*. The arrows represent the exchange of information.

Scenario 1: Set-up diagnostics connection and read emission data. A logging functionality collects information about the vehicle over time, such as the emissions data and the GPS position. In order for the data to be collected, the Repair Tool sends the emission data request signal (RED) via the Edge ECU to Engine ECU over the OBD network. The Engine ECU then sends the emission data response signal (ED), including the requested information, back to the external interactor.

Scenario 2: Extended vehicle warning Vehicle to X (V2X) communication allows the exchange of information between the vehicle and the road infrastruc-



(a) Architecture of in-vehicle communication.



(b) DFD of the architecture.

Fig. 1. The running example

ture or other vehicles. If the warning light button is active, the vehicle forwards the warning light status (WL) from the Vehicle ECU to the Driver Control. The latter sends it over the Eth 4 network to the Driver Display in order to alert the driver. The warning light status is then sent over the Eth 1 network to the Edge ECU, which, in turn, collects the current GPS position from the Connectivity Gateway. The information (both WL status and GPS position) is sent to the Edge ECU via the Driver Control ECU on networks CAN 2 and Eth 3. Finally,

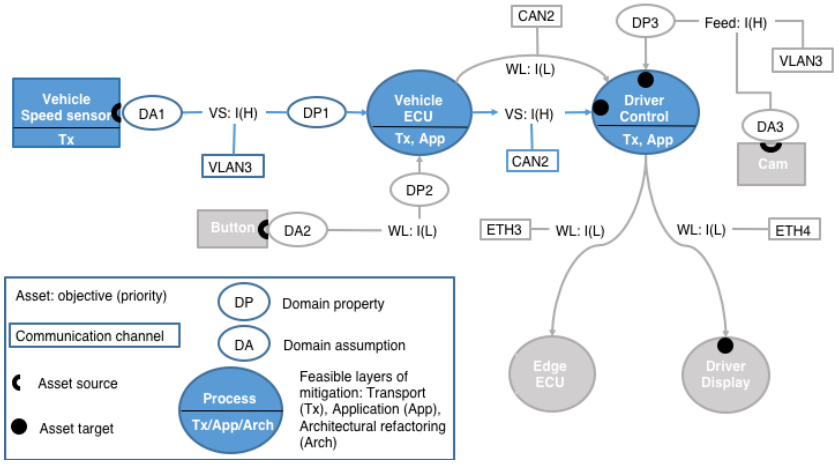


Fig. 2. The extended DFD notation for an end-to-end, asset-centric flow.

both the GPS location and the warning light status are broadcast via the V2X ECU.

Scenario 3: Using city traffic collision prevention to brake for pedestrians in the trajectory of the vehicle This scenario describes the situation where city traffic collision prevention is used to slow down or stop the vehicle if pedestrians appear in the vehicles' trajectory. The camera sensor sends live video feed to the Driver Control ECU, which analyses it for upcoming obstacles. The Driver Control ECU also receives information about the vehicle speed (VS) from the Vehicle ECU. If a possible collision is detected, the Driver Control ECU generates a vehicle speed change request (VSC) and sends it to the Engine ECU, which orderly sends a brake command (BC) to the Brake ECU.

3 An extended DFD notation

Security experts performing threat analysis are aware of the threat explosion problem and try to counter it by making abstractions. For instance, analysts often group seemingly homogeneous elements of the DFD with respect to the type of threats they are subject too. In STRIDE, this technique is called reduction [7]. To make any sort of abstractions possible, candidate elements need to be closely inspected in order to decide whether an abstraction will overall have a negative or a positive outcome. This decision should be an intelligent choice, supported by evidence in the architecture. In order to trace assets through the system, we propose to use asset-centric partial DFDs. Figure 2 shows a DFD for the *vehicle speed (VS)* asset. Notice that the DFD is a slice of the overall model presented in Figure 1b.

Security objectives and priorities. Each data flow is marked with the asset that is being transported (also part of the standard DFD notation). We

extend the notation by introducing clear markers of where an asset is generated (*asset source*) and where it is consumed (*asset target*). If assets are to be protected, they have to be analyzed from the source all the way to the target architectural element. Additionally, the asset is labeled with one or more *security objectives*: confidentiality (C), integrity (I), availability (A). Security objectives are used to focus the identification of important threats during threat analysis. Often, the analysis spans across several months with brief sessions every week or two. After a few sessions, it is easy to forget about the analysis constraints (e.g., 'focus on confidentiality only') if they are not clearly visible in the diagram. The asset is also ear-marked with a *priority* label that signifies the importance of the objective. We use the values of low (L), medium (M) and high (H). These do not express the importance of the asset as such, but rather the impact of a compromised asset objective. This information helps calibrate the depth of the analysis to come.

Security assumptions and properties (at flows). During threat analysis, assumptions are made in order to assess whether threats are feasible in the underlying architecture. For instance, the integrity of VS can be compromised if it is transported in clear and if the attacker has access to the transport medium. Together with the domain expert, the analyst may choose to make an assumption about an existing security mechanism in place that protects VS against tampering threats. When making assumptions about the system, the analysts need to be very careful not to make optimistic assumptions, which can lead to overlooked threats. A false assumption about the security of a GPS sensor can, for example, result in overlooking spoofing threats. On the other hand, not making any assumptions can make the analysis highly inefficient and result in the elicitation of irrelevant threats. Today, assumptions are sometimes still documented separately in an informal way. Considering that they are easily forgettable, they must be made visible in the model, right where they are needed.

In this paper, we distinguish between domain assumptions and domain properties, as described by Van Lamsweerde [24]. Domain properties are used to describe non disputable facts about the domain, whereas domain assumptions are statements about the domain that may or may not hold. For instance, "It is infeasible to encrypt the CAN bus" is a domain property. This is something that we can not change about the domain. On the other hand, "The CAN bus is not accessible to the attacker" is a typical domain assumption.

In Figure 2 there is one assumption and one property on top of the flow going from vehicle speed sensor to Vehicle ECU. The domain assumption DA1 is the following: "The vehicle speed sensor is a Commercial-Off-The-Shelf product and is working securely." The domain property DP1 placed on the same flow is the following: "There is a feasible mitigation solution on the transport layer for the flow between the vehicle speed sensor and the Vehicle ECU." The assumption DA1 and property DP1 are later on used to argue about the security of assets on the flow, which is discussed when abstracting the diagram. The limited layers of mitigation solutions are annotated within processes and external entities with capital letters.

Finally, there is one property that is so important (especially in embedded systems) that it gets its own annotation. We refer to the representation of the *communication channel* for each data flow. The communication channel explicitly shows which network the data flow and the corresponding asset belong to. A regular DFD notation does not include the topological behavior gathered in the communication matrix. Keeping that information visible is important, because most domain properties and assumptions that have to be made are about network and protocol capabilities.

Forward assumptions (at processes). Processes are ear-marked with this annotation, which is important in the perspective of simplifying the analysis process, as described in Section 4. In essence, we suggest that it is useful to explore the space of possible solutions which are realistic to implement in terms of mitigation mechanisms. For example, some ECUs include a Hardware Security Model (HSM), which provides transport layer encryption. This exploration needs to be done before the threat analysis starts. As the same threats start to appear more often (e.g., the assessment of the integrity of the VS is generating a lot of tampering threats along the asset flow), it is more efficient to turn a forward assumption into a domain property (i.e., mandate the adoption of a certain security mechanism, like turning on the encryption) and stop bothering about that asset all together. This kind of of backtracking in the analysis process is supported by the extended notation.

Our work differentiates between threat mitigation solutions on different layers of abstraction: transport (Tx), application (App) and architectural (Arch) layer. For instance, on the transport layer, symmetric cryptography may be used to establish a secure communication protocol between one of the sensors connected to the vehicle and the receiving ECU. On the other hand, an application layer mitigation solution would include an application layer firewall that inspects packets traveling to and from the Repair Tool in a remote diagnostic scenario. On the architectural layer, security mitigation techniques include architectural re-factoring, where possible design decisions are required to modify the system architecture. Note that middle-ware solutions, such as message queues, are also grouped as architectural layer mitigation techniques.

In addition, while the focus of abstraction is on a single end-to-end flow, elements indirectly involved in the end-to-end flow are still represented in the diagram. The reason for including additional elements is because abstractions directly effect neighbor elements. Consider what happens if two processes are folded. One part of the end-to-end flow gets successfully abstracted. However, there might have been other flows between that were unaccounted for. Neglecting the neighboring elements can increase the risk of missing important threats during the analysis. We use a different color for such elements (gray) to stress this point.

In summary, there are three important actors that participate in the process of obtaining the extended DFD. Figure 3 shows how a domain and a security expert work together with a business expert to gather the necessary information. The eDFD is built after having analyzed the problem and solution space. First,

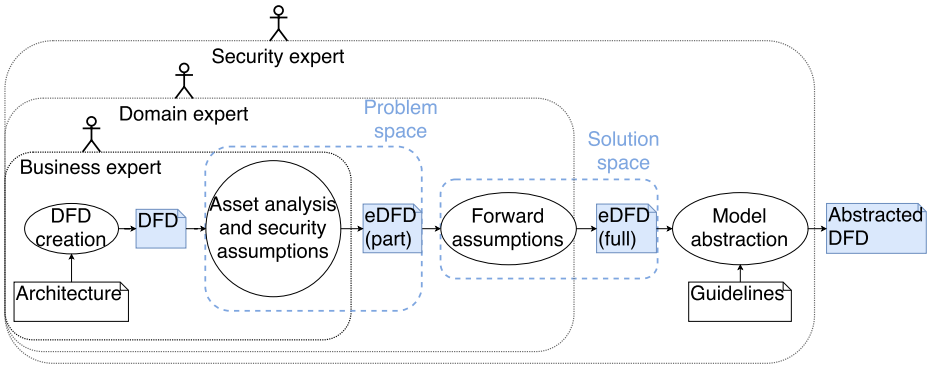


Fig. 3. Roles and responsibilities for extending the DFD using the proposed approach.

the domain expert defines an architectural model including security objectives, purpose and priority of main assets in the system. To that end, the business expert contributes with determining the priorities of assets, while the security expert helps define the objectives. The architectural model is comprised of a structural, behavioral and a topological view. This architecture is used as input to create a DFD. After the DFD defined, all three actors contribute to asset analysis, where they reason around the *problem space*. Asset analysis is performed in an iterative manner. Each asset is first identified, then the source and target components of the asset are located. Although the domain and security experts are mostly active in this phase, the business expert contributes to the discussion from a financial perspective. The asset is then traced in the DFD, where all components affiliated with the asset are marked accordingly. Asset analysis activities are repeated for all assets identified within user scenarios (behavioral view). At this point, the partially extended DFD includes security objectives, priorities and security assumptions at flows, put forward in Section 3 In order to complete the eDFD, forward assumptions about processes need to be made. When making forward assumptions about security mitigations, the actors discuss the so called *solution space*. To wit, domain and security experts limit possible mitigations in accordance with domain specific constraints and possibilities. Lastly, using the complete eDFD, the security expert makes use of the guidelines to abstract the DFD. Section 4 discusses the model abstraction activity and how it counters the threat explosion. Threat analysis begins after the DFD is abstracted.

4 Handling the threat explosion

In this section we discuss how our approach tackles the previously mentioned problem of the 'threat explosion', i.e., when too many (often irrelevant) threats are found when STRIDE is applied to a medium-to-large DFD. With the support of the running example, we illustrate how the abstraction is performed before the threat analysis and how the effort reduction is supported during the analysis.

4.1 Abstraction before threat analysis

The first approach towards solving the problem is to reduce the number of DFD elements (before the analysis starts) by means of heuristics. In such a manner the model is simplified and consequently, the analysis produces less unimportant threats. We have defined two initial guidelines for flow bundling and process folding. To this aim, the extended notation introduced previously plays an important role. The guidelines were obtained through iterative sound-boarding with industrial partners while working on the architecture from Figure 1. The reader should note that the guidelines are not meant to be a complete set of criteria, but rather the result of our initial observations. Having said that, we defined two different sets of guidelines for components with either critical or non-critical assets. For critical components, a more strict set of criteria is used, while for non-critical components the criteria are somewhat loosened. In this way, the abstraction is done in accordance with the "heat" of the system (obtained from the asset analysis) in a each region.

Bundling data flows. We consider two or more data flows (arrows in a DFD) between two processes, a process and an external entity or a process and a data store. When bundling data flows, the highest security objective of assets dictates the level of criticality. For non-critical assets, the corresponding data flows must be associated to the same communication channel (e.g., CAN 1) for the bundling to be possible. For instance, in Figure 1b the flows between the Repair Tool and Edge ECU include assets that are not critical and they are broadcast over the same network. Therefore, they can be bundled. After bundling, the resulting data flow is annotated with a new name and the union of security objectives from both bundled data flows. If the flows contain the same security objective, the highest priority is included in the union.

If any of the data flows considered for bundling contains a high security objective, the area is considered critical and additional criteria need to be met. We must look at the end-to-end flows (as in Figure 2) for the involved assets. These end-to-end flows must have either the same source, target or both. Otherwise, if they have different source and target, the unaligned parts of the end-to-end flows should not be critical. For instance, in Figure 2 the data flow VS between Vehicle ECU and Driver Control ECU is marked with a high-priority integrity objective, while WL has a low-priority integrity objective. Therefore, this area in the architecture is considered critical. Both VS and WL data flows are broadcast on the same communication channel. The diagram shows that the VS end-to-end flows goes from the Speed Sensor to the Driver Control. The WL signal end-to-end flows goes from the Warning Light Button (pressed by the driver) to the Driver Display. Therefore, the assets VS and WL do not have the same source or target. Rather, they only align between the Vehicle ECU and Driver Control. However, the criticality of the non aligned parts is low. In conclusion, the data flows can be bundled according to the more restrictive criteria.

Process folding. Candidate elements for process folding are two adjacent processes. All flows between the candidate processes are considered when determining the criticality of the region.

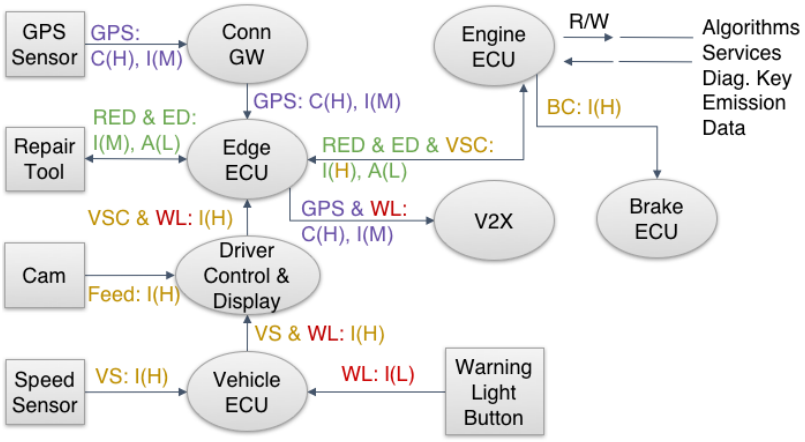


Fig. 4. The abstracted DFD after applying the guidelines.

If the flows do not transport high-priority assets, the region is considered as non-critical. Non-critical processes may be folded if (a) they are not near to a trust boundary and (b) there is a mitigation in place (security assumption) that ensures that at least one of the objectives for the surrounding flows is covered. If a trust boundary is next to the considered region, the processes are likely part of the attack surface and it is considered too risky to bundle them, as relevant threats might be overlooked. In the example of Figure 2, this guideline applies to the region containing the Driver Control and Driver Display: there are no high priority assets broadcast on ETH 4 and the processes are not near to a trust boundary. Hence, the two processes can be folded.

If any flow between the candidate processes contains a high security objective, the criteria are more restrictive. In addition to the above-mentioned conditions, the processes (c) need to be "mounted" on the same communication channels and (d) there must be mitigations in place ensuring that *all* security objectives over the flows are covered. Unfortunately, no such fold was possible in the running example.

Figure 4 presents the results of the abstraction obtained by applying the guidelines described in this section. Overall, the difference between the original DFD (in Figure 1b) and the abstracted DFD (in Figure 4) lies in the number of flows and processes. The abstracted DFD has 1 process less (Driver Display) and 7 flows less compared to the initial DFD. The simplifications to the model result in a reduced number of identified unimportant threats, as further discussed in Section 4.3.

4.2 Effort reduction during threat analysis

Abstracting the architectural model before threat analysis can only take the analyst so far. It is not only the number of DFD elements that makes threat analysis time consuming, but also the type and amount of threat patterns that must be considered for each element. However, the additional information about the assets and the forward assumptions may also be used *during* the analysis to guide the analyst towards the important threats, while omitting the not important ones.

For instance, in Figure 4, the RED and ED flows have been bundled as a result of the abstraction step. The assets on this bundled flow are ear-marked with the integrity objective (medium priority) and the availability objective (low priority). Therefore, the analyst can focus on the tampering and denial of service threats, and ignore the information disclosure threats[4]. As the priorities are not high, the analyst also knows that it is not necessary to dig too deep in the analysis process of this flow. In this respect, the analyst can bring these risk considerations into the process of threat identification and leverage them to reduce the effort spent analyzing this "cool" spot of the system. As a result of the analysis, we found two important threats on that particular flow: physically damaging the OBD port and tampering with the ED signal before it is displayed on the Repair Tool.

Another means of reducing the effort spent eliciting the threats is to use the security assumptions during the threat analysis. For instance, let us suppose we are analyzing the vehicle speed (VS) asset-centric flow described in Figure 2. The asset is ear-marked with a high-priority integrity objective and, hence, the analyst should carefully consider the potential tampering threats. However, Because of the domain assumption DA1 ("The speed sensor is working securely") and the domain property DP1 ("Transport-layer security is used between the ECUs"), the tampering threats can be discarded altogether as they are non interesting.

4.3 Effect of abstraction

The resulting abstracted DFD has been analyzed (by the first author) using the off-the-shelf threat catalogs of STRIDE. This led to the identification of 15 threats, which are not presented here due to space constraints. To appreciate the efficiency of our approach, we remark that a previous study [18] has shown that according to the traditional STRIDE approach, the number of identified threats from a DFD this size should have been around 100. The 15 identified threats have been reviewed by a security expert (the third author) who routinely performs the threat analysis of automotive systems. The validity and relevance of the identified threats have been confirmed by the expert. Further the expert confirmed that no relevant threats had gone unnoticed.

Our approach resulted in finding less threats because of two reasons. Processes and flows are the elements that are the main cause of threat explosion

[4] According to STRIDE, a data flow is subject to three types of threats: tampering (T), information disclosure (I), and denial of service (D).

in a DFD. First, together they make up a large number of architectural elements. Second, they are prone to many types of attacks and, hence, have to be analyzed for several threat categories. Out of a total of six threat categories, STRIDE mandates the consideration of three categories for flows and of all six categories for processes. It is apparent that reducing the number of processes and flows in the DFD can help govern the threat explosion problem. Second, a further reduction of effort is due to a more focused analysis, as explained in Section 4.2.

5 Related work

Significant work has been done in the area of threat analysis and risk assessment (TARA) methods in the automotive domain. Macher et al. [11] recently performed a review of TARA methods in the automotive context. In their main findings, the authors identify most applicable TARA methods for early phase analysis, which closely relate to our approach. The EVITA [2] method is an adaptation of the ISO 26262 HAZOP analysis for security engineering. The method considers potential threats for particular features from the functional perspective by developing attack trees and eventually discovering worse case scenarios. Even though the method employs leveled qualitative risk assessment, no effort is mentioned regarding attack tree minimization. HEAVENS security model [3] analyzes threats based on the STRIDE threat modeling approach and ranks them by assessing the risk with determining the threat, the impact and finally the security level. In contrast to our approach, HEAVENS model is oriented towards ranking the threats only after identifying them. SAHARA [12] method combines the automotive HARA safety analysis with the STRIDE approach to discover impacts of security threats in the safety analysis. The focus of SAHARA lies in understanding the relationship between security threats and safety implications, whereas our work focuses on security aspects only. A security analysis of several applications within a Connected Vehicle Reference Architecture (CVRIA) has been performed by ITERIS and is available online [1]. The published documents include a CIA asset analysis of the V2X communications, while our work focuses on the in-vehicle communications.

Beyond the domain of automotive software, other asset- or software-centric threat modeling approaches are relevant to our work. STRIDE [20] is a popular threat modeling approach, which is based on DFDs. The methodology is comprised of 7 steps: define users and realistic use scenarios, gather assumptions, construct a DFD diagram of the system, map STRIDE to DFD element types, refine threats, document the threats, assign priority via risk analysis (to counter threat explosion problem) and select mitigations associated to threats. In STRIDE, threat prioritization according to risk value is done after the threat elicitation. Additionally, although STRIDE suggests to start by gathering the security assumptions, no explicit guidance is provided on how to represent and use them in the threat analysis process. Similar to STRIDE, TRIKE [16] is a software-centric methodology. TRIKE includes the identification of assets and

actors and offers tool support for attack tree and graph generation. CORAS [10] is an asset-centric methodology for risk analysis consists of a language, a tool and a method. The methodology employs CORAS threat diagrams that describe the threats, vulnerabilities, scenarios and incidents of relevance for the risk in question. Similarly to the aforementioned CORAS methodology, PASTA [23] is an asset-centric, risk-based threat modeling methodology. In PASTA threats are analyzed with the help of use cases and Data Flow Diagrams (DFDs). Further steps are taken for detailed analysis, namely the use of attack trees and abuse cases.

Looking towards recent initiatives to automate threat modeling, our approach relates to the work on extracting threats from DFDs by J. Berger et al. [6]. Similarly to our work, the authors introduce additional semantics, including the topological behavior. Furthermore, they also develop a set of guidelines, which are used to build a threat model of the architecture. However, these rules are used to discover only cataloged threats and do not aim to handle threat explosion. Perhaps more importantly, our approach differs by analyzing end-to-end assets which, in turn, drives the model abstraction and threat reduction.

Interesting work has been done by Rauter et al. [14] in developing a metric that quantifies software components by their ability to access assets. By doing so, the authors are also able to identify critical areas in the software architecture and consider those for a detailed threat analysis. However, they do not discuss the specifics of how threat analysis techniques benefit from their architectural risk assessment method. Our work also relates to the automated software architecture risk analysis studied by Almorsy et al. [5]. The introduced approach is accompanied by a tool and explores security risk analysis by means of formalized signatures of security scenarios and metrics. Similarly, the authors develop a risk-centric architectural analysis approach. However, their work focuses on operationalizing attacks and security metrics to assess the risk level, instead of aiming towards systematically identifying important threats. In addition, the formalized signatures do not seem to consider end-to-end flows of assets.

We also identify several related approaches that could be grouped as attack-centric threat analysis techniques. In these approaches, an explicit model of the attacker is introduced and the analysis is performed from the perspective of an attacker. Examples of such techniques are anti-goals [8], misuse cases [21], misuse case maps [22], abuse cases [13], abuse frames [9], and attack trees [19, 15].

6 Discussion and limitations

The process of creating the enriched model described in Section 3 results in a deeper understanding of the system before threat analysis activities take place. Not only does this contribute towards a common security awareness, but it also enables the identification of realistic threats in the system. As previously mentioned, one of the drawbacks of STRIDE is that the analysis of DFD elements is performed in isolation. In contrast, attack strategies target an asset and often affect a combination of elements. Our approach implicitly considers all the model

elements that are related to an asset, which contributes towards detecting attack strategies earlier on in the software development life-cycle. Most importantly, our approach is a step further towards optimizing the effort spent on analyzing threats. This aspect is of primary importance in the industry (especially for complex systems, like CPS) and is often overlooked in the related work. Even though the guidelines are only initial observations, they are synthesized in a domain-independent way, where the main role is played by the semantics of end-to-end flows and not the domain-specific content. Therefore, there is potential for generalizing the guidelines to domains outside the scope of cyber-physical systems. In particular, domains where the software architecture is comprised of different networks and communication protocols may benefit from our approach (e.g., Microservice architecture). However, more investigations have to be made in order to confirm this claim.

Our approach relies on the correctness of the domain assumptions and the truthful representation of the domain properties. This means that the presence of a domain expert is mandatory. Another draw back is that the approach assumes that there are in fact non-critical areas in the architecture. If all the candidate model elements for abstraction have high-priority security objectives, the abstraction may result fewer simplifications, if any at all. Another problem might arise once the amount of end-to-end flows increases. The guidelines do not consider what happens when new scenarios are added. New scenarios might bring along different assets that travel through the same communication channels. As a result, successful abstractions have to be reconsidered. Some of these problems could potentially be alleviated by a tool. In terms of increasing the productivity, the application of guidelines for abstraction can be (semi)automated. Such support may take advantage of our approach, while enabling experts to freely move between abstractions during the analysis. Working towards the automation of threat analysis also caters to security related activities later in the product life-cycle. Notably, after implementation, the planned architecture comes seldom into question again. However, the implemented architecture often differs from the planned architecture too much. Checking for compliance is therefore a complex and important task. As part of future work, we plan to explore ways to synchronize the extended DFDs and the implemented architecture. Furthermore, we acknowledge that we have validated the approach only by illustrating its potential on one simplified architecture. In future work, we will systematically evaluate the benefits of our approach in a series of comparative studies involving both students and industrial experts.

7 Conclusion

In this work, we presented a novel approach for a risk-first security analysis of design artifacts. Without departing too much from well-known techniques like STRIDE, our approach is focused on an end-to-end asset analysis and accommodates forward reasoning on the constraints imposed by the solution space. Our main contributions are (i) a notation to represent end-to-end asset flows

and to enrich the analysis models with important security assumptions, and (ii) a set of guidelines for traversing the model and making suitable abstractions. The contributions aim at mitigating the problem of threat explosion, commonly present in STRIDE and other techniques. Additionally, the extended notation supports a more appropriate representation of communication channels, which is valued in the domain of Cyber-Physical Systems. We illustrated our approach by applying it on a simplified system provided by industrial partners in the automotive domain. Preliminary results show that the analysis method indeed yields a reduced number of low priority threats. In future work, we plan to extend the validation of our approach and explore the opportunities for threat analysis automation.

Acknowledgments

This research was partially supported by the Swedish VINNOVA FFI project "HoliSec: Holistic Approach to Improve Data Security".

References

1. Connected vehicle reference implementation architecture. http://local.iteris.com/cvria/, accessed: 2017-8-25
2. E-safety vehicle intrusion protected applications. http://www.evita-project.org/index.html, accessed: 2016-11-25
3. Heavens: Healing vulnerabilities to enhance software security and safety. http://www.vinnova.se/sv/Resultat/Projekt/Effekta/HEAVENS-HEAling-Vulnerabilities-to-ENhance-Software-Security-and-Safety/, accessed: 2016-11-25
4. Holisec: Holistiskt angreppsstt att frbttra dataskerhet. http://www2.vinnova.se/sv/Resultat/Projekt/Effekta/2009-02186/HoliSec-Holistiskt-angreppssatt-att-forbattra-datasakerhet/, accessed: 2017-06-14
5. Almorsy, M., Grundy, J., Ibrahim, A.S.: Automated software architecture security risk analysis using formalized signatures. In: Proceedings of the 2013 International Conference on Software Engineering. pp. 662–671. IEEE Press (2013)
6. Berger, B.J., Sohr, K., Koschke, R.: Automatically extracting threats from extended data flow diagrams. In: International Symposium on Engineering Secure Software and Systems. pp. 56–71. Springer (2016)
7. Howard, M., Lipner, S.: The security development lifecycle, vol. 8. Microsoft Press Redmond (2006)
8. van Lamsweerde, A.: Elaborating security requirements by construction of intentional anti-models. In: Proceedings of the 26th International Conference on Software Engineering. pp. 148–157. ICSE '04, IEEE Computer Society, Washington, DC, USA (2004), http://dl.acm.org/citation.cfm?id=998675.999421
9. Lin, L., Nuseibeh, B., Ince, D., Jackson, M.: Using abuse frames to bound the scope of security problems. In: Requirements Engineering Conference, 2004. Proceedings. 12th IEEE International. pp. 354–355. IEEE (2004)
10. Lund, M.S., Solhaug, B., Stølen, K.: Model-driven risk analysis: the CORAS approach. Springer Science & Business Media (2010)

11. Macher, G., Armengaud, E., Brenner, E., Kreiner, C.: A review of threat analysis and risk assessment methods in the automotive context. In: International Conference on Computer Safety, Reliability, and Security. pp. 130–141. Springer (2016)
12. Macher, G., Sporer, H., Berlach, R., Armengaud, E., Kreiner, C.: Sahara: a security-aware hazard and risk analysis method. In: 2015 Design, Automation & Test in Europe Conference & Exhibition (DATE). pp. 621–624. IEEE (2015)
13. McDermott, J., Fox, C.: Using abuse case models for security requirements analysis. In: Computer Security Applications Conference, 1999.(ACSAC'99) Proceedings. 15th Annual. pp. 55–64. IEEE (1999)
14. Rauter, T., Kajtazovic, N., Kreiner, C.: Asset-centric security risk assessment of software components. In: 2nd International Workshop on MILS: Architecture and Assurance for Secure Systems (2016)
15. Saini, V., Duan, Q., Paruchuri, V.: Threat modeling using attack trees. Journal of Computing Sciences in Colleges 23(4), 124–131 (2008)
16. Saitta, P., Larcom, B., Eddington, M.: Trike v. 1 methodology document [draft]. URL: http://dymaxion. org/trike/Trike_v1_Methodology_Documentdraft. pdf (2005)
17. Scandariato, R., Walden, J., Joosen, W.: Static analysis versus penetration testing: A controlled experiment. In: Software Reliability Engineering (ISSRE), 2013 IEEE 24th International Symposium on. pp. 451–460. IEEE (2013)
18. Scandariato, R., Wuyts, K., Joosen, W.: A descriptive study of microsoft's threat modeling technique. Requirements Engineering 20(2) (2015)
19. Schneier, B.: Attack trees. Dr Dobb's Journal, v.24, n.12 (1999)
20. Shostack, A.: Threat modeling: Designing for security. Wiley (2014)
21. Sindre, G., Opdahl, A.L.: Eliciting security requirements with misuse cases. Requirements Engineering 10(1), 34–44 (2005), http://dx.doi.org/10.1007/s00766-004-0194-4
22. Tøndel, I.A., Jensen, J., Røstad, L.: Combining misuse cases with attack trees and security activity models. In: Availability, Reliability, and Security, 2010. ARES'10 International Conference on. pp. 438–445. IEEE (2010)
23. UcedaVelez, T., Morana, M.M.: Risk Centric Threat Modeling: Process for Attack Simulation and Threat Analysis. John Wiley & Sons (2015)
24. Van Lamsweerde, A.: Requirements engineering: From system goals to UML models to software, vol. 10. Chichester, UK: John Wiley & Sons (2009)
25. Wuyts, K., Scandariato, R., Joosen, W.: Empirical evaluation of a privacy-focused threat modeling methodology. Journal of Systems and Software 96, 122–138 (2014)
26. Yu, H., Lin, C.W.: Security concerns for automotive communication and software architecture. In: Computer Communications Workshops (INFOCOM WKSHPS), 2016 IEEE Conference on. pp. 600–603. IEEE (2016)