Towards the Model-Driven Engineering of Security Requirements for Embedded Systems

Yves Roudier, Muhammad Sabir Idrees
EURECOM
Network and Security Department
Sophia Antipolis, France
{Yves.Roudier | Muhammad-Sabir.Idrees}@eurecom.fr

Ludovic Apvrille
Institut Mines-Telecom, Telecom ParisTech
CNRS/LTCI, System-On-Chip laboratory (LabSoC)
Sophia Antipolis, France
ludovic.apvrille@telecom-paristech.fr

***Abstract*—This paper discusses why and how security requirements engineering must be adapted to the model-driven approach usually adopted to design and develop embedded systems. In particular, we discuss to what extent the elicitation of security requirements and the Y-chart partitioning approach, a central design methodology in embedded systems, can mutually enrich each other. We also show how SysML, which is already commonly used to engineer requirements in embedded systems, can also represent security requirements, assets, and threats with only a few extensions and thus support a more comprehensive requirements engineering methodology. We illustrate the use of our overall methodology and toolkit with examples from the automotive embedded system field in order to demonstrate the relevance of our approach.**

***Index Terms*—Communication System Security; Computer Security; Design methodology; System-level design; Component Architectures; Embedded Systems; Security Requirements Engineering**

I. Introduction

Embedded systems are pervading our daily experience of technology in all sorts of devices, appliances, or command and control systems. Embedded systems are made of both software and hardware electronic components that are tightly coupled together to form the embedded system architecture. The software part of those systems offers flexibility to products after they are released to the market, for instance through firmware updates - something impossible with plain hardware. Many of those systems exhibit a heterogeneous and distributed architecture, featuring multiple computational units (microprocessors or micro-controllers) interconnected using communication buses.

The security of embedded systems is becoming a tremendous problem in our technology-powered environment. Many security mechanisms have been developed for such systems, but their use generally disregards functional and other non-functional requirements expressed, and they are often introduced late in the design cycle. Still, the already many security requirements analysis methodologies do not integrate with the usual workflow of embedded system architects. This paper discusses why the model-driven engineering techniques used to design and develop the embedded systems, and in particular the way they capture the hardware/software partitioning and security requirements engineering should go hand in hand. We define a new SysML-based framework, called SysML-Sec, to support the engineering of security requirements for embedded systems. SysML is commonly used to capture embedded system architectures and functional requirements. We explain how both assets and complex sequences of attacks can also be represented using only a very limited extension of SysML so as to fit the model-driven design process of embedded systems.

II. Motivation and Contributions

The security requirement and threat analysis is mostly considered as a preamble to risk analysis in IT systems. This process is generally meant to decide whether to introduce security countermeasures into the system, which results in additional costs. In the case of embedded systems, we contend that the security analysis also has a strong impact on the design of the system architecture. This is especially true in safety-critical systems, where attacks may be devastating, but where security functions overhead may also result in an absolutely useless system. We also claim that the security analysis should also play an important role with respect to convincing the designer of increasingly complex embedded systems of the consistency and exhaustivity of his security architecture, at least with respect to the threats identified.

A. Security Attacks over Embedded Systems

Security issues have received too little attention in the context of embedded systems, contrary to the situation in IT systems. However, an increasing number of embedded systems have become communicating artifacts, feature new interactions with their immediate environment or with backend systems, and are thus exposed to criminals. The following examples illustrate typical attacks and security concerns:

Set-Top Boxes. The attack of Microsoft's XBox [1] is a famous case of a physical attack on an unprotected bus. Other types of set-top boxes have also been targetted with application software or protocol stack attacks, like for instance ADSL routers [2].

Mobile Appliances. Mobile appliances, like typically mobile phones, come with many security challenges with respect to existing vulnerabilities. Attacks are getting increasingly complex, e.g., hackers trying to achieve iOS jailbreaks, commonly involving the use of several vulnerabilities in chain [3].

Avionics. Modern aircrafts feature a handful of embedded systems, each concerned with a specific mission. Airplane internal networks are now interconnected with ground systems as well as with passenger or attendant networks and thus raise serious concerns that require the introduction of filtering mechanisms. Recent studies on navigation systems have repeatedly reported the lack of any attention to security in their design as well as the risks and means of possible exploitation of those lacks [4], [5].

Automotive Systems. A modern automotive on-board network interconnects a hundred of microcontrollers, termed Electronic Control Units (ECUs) organized into application-specific domains bridge by gateways. Attacks have been shown to be quite feasible [6] by bypassing the filtering performed between domains or by brute-forcing ECU cryptography-based protection mechanisms. Such attacks may in practice originate from the Internet connection increasingly available in vehicles or even from the Bluetooth pairing of a compromised mobile phone to the vehicle on-board network. Further attacks are anticipated in upcoming Car2X applications, which will feature vehicle-to-vehicle or vehicle-to-infrastructure communications.

B. Contributions

Our work first aims at introducing a security analysis into design approaches currently used for embedded systems, and to ensure that the design converges towards a workable solution. This solution should notably mitigate relevant threats with proper security mechanisms, yet ensure that those mechanisms do not prevent a correct operation of the system due to insufficient communication resources (too much overhead from security mechanisms, in particular cryptographic protocols) and execution resources (too much overhead from security computations under real-time assumptions, e.g. verifying signatures of all messages between two microcontrollers). This is inherently an iterative process that must be integrated into existing design tools and methodologies. For instance, attacks on the bus between a CPU and its memory can only be identified once the existence of the bus and the messages exchanged over it are decided by the system designer and there should be a way for the security expert and the system designer to communicate their respective concerns using a unified framework.

Our work secondly aims at defining mechanisms to support the modeling of threats and related counter-measures. That modeling should be useful at the multiple levels of abstraction that one has to consider in an embedded systems. For instance one may describe attacks on sets of physically interconnected hardware components or on sets of logical functions mapped onto them (such zones might even be defined within a CPU). Technologies like virtualization and trusted computing, best illustrated by the Trusted Computing Group's TPM or ARM's Trusted Zones, can prevent attackers from trespassing on such a zone. Tamper-resistant hardware, like for instance a smartcard, is also commonly used to protect either secrets or computations. We know of no security methodology describing such trusted perimeters and mapping threats to them.

We finally also intend to provide a framework to support composition through documenting component vulnerabilities. The reuse of components in a systems product line portfolio pleads for the capitalization of such knowledge so as to better assess potential risks when several software and hardware components are combined together. The framework we propose in the rest of the paper also aims at supporting this activity. However, we do not address risk assessment in the scope of this paper. We consider that our proposal can be made compatible with several existing proposals in this domain, including the one we developed and used in the EVITA case study [7], [8].

C. Use Cases

Throughout this paper, and without any loss of generality, we will give examples of the use of our methodology, SysML-Sec, with results from a study that we conducted in the scope of the FP7 European research project EVITA. The project, which included a major car manufacturer and tier-one equipment suppliers, investigated the security of automotive on-board networks. A prototype secure automotive on-board network was designed and developed by the EVITA partners.

Two scenarios - out of around 20 - are particularly used to illustrate our requirements engineering methodology. The first one, *Firmware Flashing*, deals with maintenance operations at the workshop. It notably addresses the need to update the firmware of an ECU in a secure manner that should protect both the intellectual property of the car maker or equipment supplier and the integrity of the update process. The second use case, *Local Danger Warning*, deals with preventing attacks on safety-critical functions of the car for displaying warnings from internal sensors or from nearby stakeholders (road-side units or other vehicles). The architecture developed should notably prevent the display of forged warnings, like for instance a bogus notification about an accident.

III. Extending Embedded Design Methodologies with Security Requirements Engineering

Safety-oriented Model-Driven Engineering methodologies for embedded systems are now well established, since many embedded systems are safety-critical, i.e., their malfunction may harm their users. Yet, the assessment of the security of embedded systems and of its impact on their safety is not.

A. MDE and Embedded Systems: Current Practice

OMG's Model Driven Architecture defines two abstraction levels, namely the Platform-Independent Model (PIM) and Platform Specific Model (PSM): embedded systems are thus clearly targeted by MDE. The SPT [9] and MARTE [10] profiles have also been defined by the OMG to more specifically address embedded systems but none of them addresses requirements modeling. Conversely, the SysML OMG profile [11] clearly takes into account requirements with explicit modeling

capabilities and diagrams, but addresses the partitioning phase only to a very limited extend.

The existence and use of specific design practice in the embedded system field make it compelling to develop an adapted security requirements engineering methodology that might be integrated to the designer's toolbox. We decided to rely on SysML for Model-Driven Engineering because of its increasingly large adoption and of its large scope of application. The MARTE approach could be used instead for the partitioning.

B. The Importance of the Hardware / Software Partitioning

Software-centric systems are commonly designed with a V-cycle, with building stages (requirements, analysis, design, implementation) followed by verification stages (e.g., tests, formal proof). In the case of embedded systems, the V-cycle can obviously start only once functions have been partitioned in software and hardware.

System partitioning is a process to analyze various functionally equivalent implementations of a system's specification. It usually relies on the Y-chart approach [12] depicted in Figure 1:

1) *Applications* are first described as abstract communicating tasks: tasks combine several related functions. Tasks and functions are defined independently from their implementation.
2) Targeted *architectures* are independently described from tasks. They are usually described with a set of abstract execution nodes (e.g., CPU with operating systems and middleware, hardware accelerators), communication nodes (e.g., buses), and storage nodes (e.g., memories).
3) A *mapping* process defines how application tasks and abstract communications between tasks are assigned to computation and communication / storage devices, respectively. For example, a task mapped on a hardware accelerator is a hardware-implemented function whereas a task mapped over a CPU is a software implemented function.

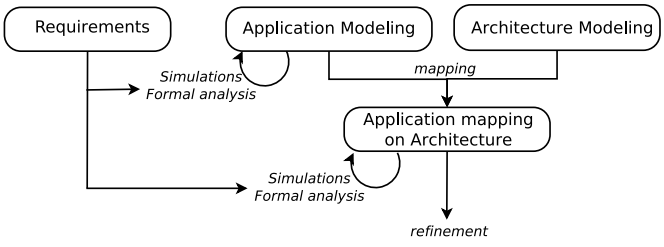

Fig. 1. The Y-Chart Methodology

The result of this process shall be an optimal hardware / software architecture with regards to criteria at stake for that particular system (e.g., cost, area, power, performance, flexibility, etc.). This partitioning process is of utmost importance. Indeed, if critical high-level design choices are invalidated afterwards because of late discovery of issues (performance, power, etc.), then it may induce prohibitive re-engineering costs and late market availability. Thus, we contend that an exhaustive identification of security requirements and the related threat analysis has to be iterated together with the partitioning.

An example of function mapping is given in Figure 2, in the scope of the Local Danger Warning use case. Two ECU sub-domains are represented. The *Chassis Safety Controller* on the left, and the *Body Electronic Module* on the right. Each sub-domain has a main processor, a local flash memory, a local main memory, a set of hardware accelerators, and a bridge to the main system bus. Functions are mapped either on processors (these functions are to be software-implemented) or on hardware accelerators (functions are to be hardware-coded). Communications between tasks are also to be mapped over buses and memories, in order to highlight data transfers.

C. SysML-Sec: a Security-Aware Model-Driven Methodology

We introduce the SysML-Sec framework, which aims at guiding and providing representations of security requirements compatible with the MDE approach of practitioners that are not security specialists. We in particular expect security experts to collaborate with system engineers through this means.

Our methodology, part of the SysML-Sec framework, captures the design objectives described in the previous sections flexibly with the evolution of the architecture specification. We can summarize its different phases with the following iterative process:

- **Initial Architecture Mapping.** One or several typical use cases are selected as a starting point. The functionalities of the system highlighted in these use cases are first modeled as tasks. Communications between tasks are modeled with information and event flows. Event-based communications is also captured in order to control the Information Flow. Tasks and communications can then be mapped to a draft architecture of the system. The designer's experience plays a key role for determining first draft architectures.
- **System Assets Identification.** System assets are identified among architectural elements (processors, pieces of software, communication channel) and will first refer to generic components, like for example: "all system buses". When the architecture gets more detailed, assets are more likely to be refined into specific elements.
- **Threats Identification and Assessment.** Threats and security vulnerabilities of the selected assets are documented using our extensions of SysML parametric diagrams. Threats should as much as possible describe the capabilities that an attacker should meet or exceed, as well as information about the origin of attacks (local, remote, through a specific interface). We also implemented automated checks of the threat coverage by security goals. Based on a risk analysis (which is not addressed in this paper), one should also identify and prioritize security goals that are mapped to a threat. The most important security requirements should be further refined. Other

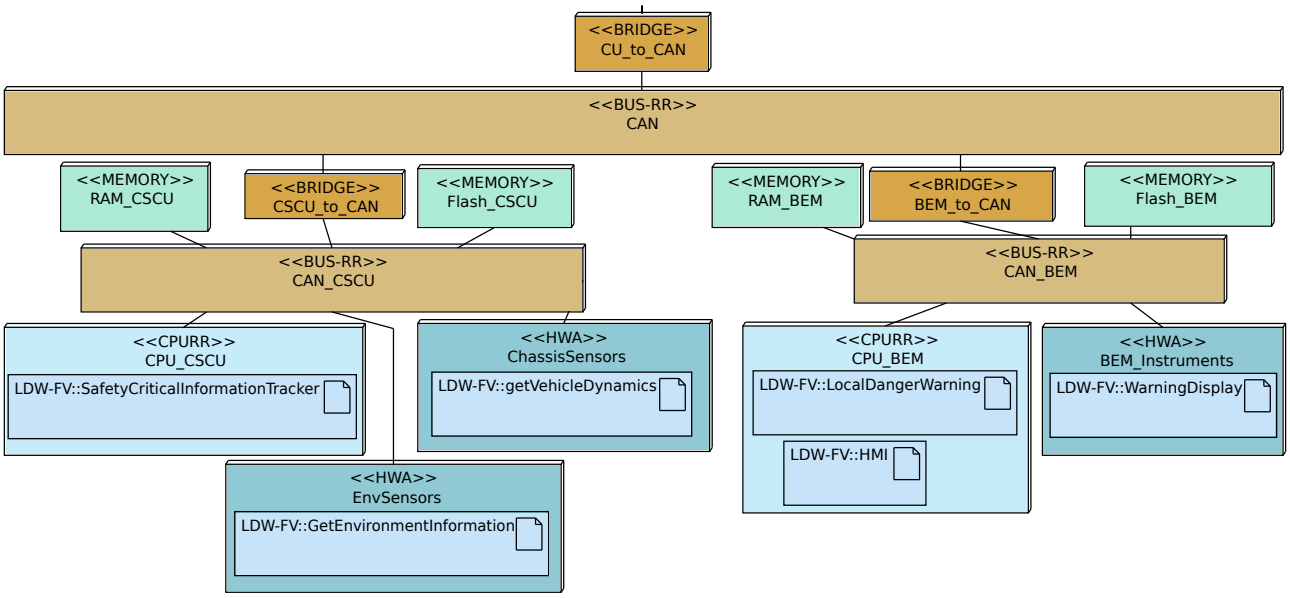

Fig. 2. Mapping of the *Local Danger Warning* Use Case (SysML-Sec)

requirements may be left aside or even abandoned at this stage, yet are remembered in SysML diagrams, especially in case an alternative analysis is required.

- **Security Goals Identification.** Security goals might be identified (1) from the above use cases, for instance because of imposed standards or of the properties expected from the system, or (2) from unaddressed threats or attacks on assets, or (3) as the refinement of another security objective when the process is iterated and the level of detail of the architecture has changed. They are represented using SysML requirement diagrams. In further iterations, one may need to delete or adapt security goals deprecated by modifications on the architecture.
- **Architecture Refinement.** The architecture, including the specification of assets, is refined. This refinement will result on the one hand in a more detailed description of the architecture components as use cases or the architecture are becoming more precise (e.g., new communication channels, refinement of an execution environment into OS/middleware/application layers, etc.). On the other hand, it may also result in the linking of requirements to system information flows. Finally, the ongoing process is iterated to the identification of new security concerns. The refinement phase may fail if the architecture and security requirements are incompatible, for instance, if the performance overhead of security mechanisms is too high. Consistency checks should also be performed to ensure that a security objective does not conflict with another requirement expressed over the same asset. A failure is the sign that the analysis should be backtracked to the previous level of detail, and possibly that other security solutions should be adopted. Such a backtrack in the methodology might be supported by the versioning of SysML-Sec diagrams.

D. SysML-Sec: Profiles and Tool Support

The aforementioned methodology has been implemented with TTool [13]. TTool is an open-source toolkit that supports several UML profiles, including DIPLODOCUS [14], and AVATAR [15]. TTool includes diagramming facilities, formal and non-formal code generators, model simulators, model-checkers and analysis tools. The main strength of TTool is to hide knowledge of the underlying simulation or formal proofs techniques, thus offering a press-button approach to perform safety or security proofs.

SysML-Sec relies on DIPLODOCUS for the partitioning phase. In particular, security requirements and attacks strongly related to hardware elements (e.g., bus probing, memory dump, etc.) can be more easily identified, and sometimes evaluated. The impact of security mechanisms can also be tested and formally verified, e.g., studying whether a given safety-related deadline may be violated because of security functions. SysML-Sec has been built over AVATAR for requirements elicitation, threats capture and security mechanisms design. Both safety and security related properties can be captured, and further proved. Our tool also provides some basic automation, like the verification of the coverage of threats with appropriate security requirements (or absence thereof), as depicted in Figure 3.

IV. Identifying and Analyzing Security Requirements Based on the Architecture

A. Security Goals and Requirements

Haley et al. [16] have shown that the meaning of a security requirement significantly varies in the literature. As they point out, security requirements must be precise enough and should make it possible to describe what objects we need to protect and why, and to describe how architectural vulnerabilities are addressed. Haley et al. conclude that one should follow a

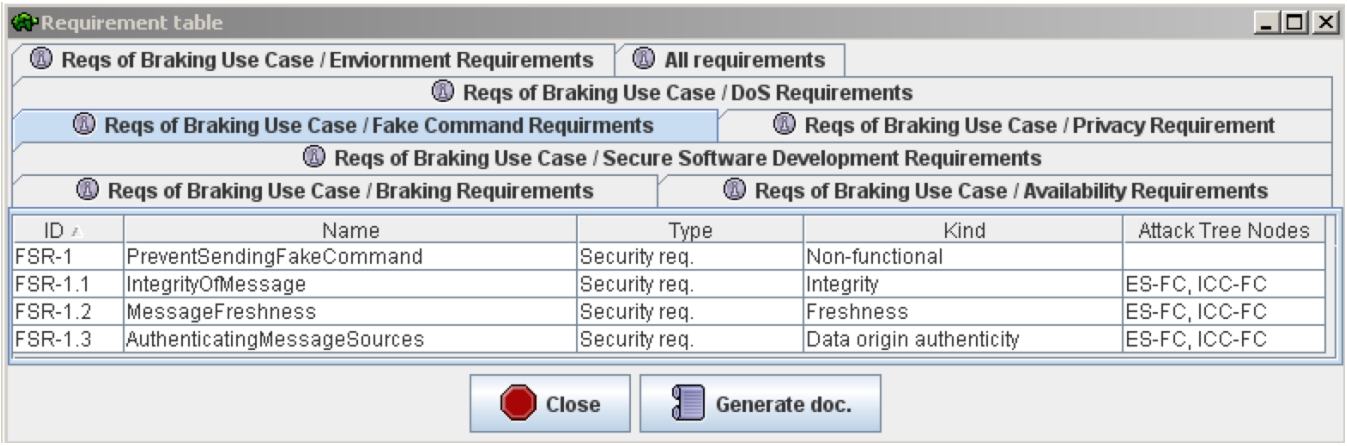

Fig. 3. Security Requirements Coverage Table for Automated Consistency Checks in TTool

goal-based approach together with the description of a context which is strongly connected with the execution environment. They distinguish between security goals and requirements as follows: the latter "express the systems security goals in operational terms, precise enough to be given to a designer/architect" and provide "a specification". Though we mostly agree with this point of view, we do not draw such a clear-cut line in the case of our representation. For example, some precise security requirements can be exhibited at very early stages of design because they are quite generic. In contrast, some security goals cannot be elicited before the architecture and its partitioning candidates have been refined to a lower abstraction level. What we considered as seemingly precise security requirements also sometimes became only complex security goals during our case study. We therefore describe security goals and requirements indistinctly as SysML requirements, even though they may be explicitly stereotyped as goals and requirements to distinguish their level of abstraction. The SysML requirement attribute *Kind* together the number of dependencies relationships with other requirements also help to precise their level of abstraction.

B. Security Assets

Haley et al. [16] further describe assets as a central notion behind the definition of security goals. In particular, finding an asset leads to envisioning threats to it, and choosing particular security principles to apply to secure it. We agree with this perspective, though we suggest to assess the type of security asset considered, which will affect the nature of related security requirements in an embedded system. In particular, we classify assets into three categories: (1) hardware components (processors, memories, communication channels, clocks, ...), including the data (and secrets) they contain; (2) software assets (virtualization mechanisms, drivers, protocol stacks, applications, ...); (3) information and event flows, which capture communication: those flows reflect the composition of the functions used to realize a particular service together with communication functions to transfer their data and their control events. Such a flow can typically be represented on a SysML block diagram, where each block corresponds to a system function and each link between two blocks corresponds either to a event or data flow. Figure 4 illustrates for instance a subset of the functional view - at the partitioning stage - of the flow that would lead a car to receive or forward emergency situations in the EVITA system. Relationships between those elements (assets, attacks, and requirements) will be defined by a model as we describe farther below.

C. Security Threats and the Architecture

Security threats define situations that may lead to a system failure, that is, a successful attack, in the presence of a malicious behavior. We are interested in describing which assets must be compromised by an attacker in order to perform a successful attack. The identification of a security threat not covered by any security goal over the targeted security assets clearly points at an incomplete elicitation of security requirements. The designer typically has to either reconsider the mapping of the security goal to assets, as described further below, or to identify missing or incomplete security goals. On the other hand, while the identification of threats is worthwhile for validating security goals, we do not expect them to be exhaustively identified. Security requirements might simply arise from the need to be interoperable with a certain security standard or even out of the certification of the system against a protection profile, as defined for instance in the Common Criteria scheme.

Attacks consist of a list of actions over assets (read a message on a bus, etc.). Threats on embedded systems now commonly consist in the exploitation of, not just one, but a series of vulnerabilities of system components as well as errors in the design of the system. Threats are frequently modeled with a wide variety of attack trees. We represent a similar concept in a SysML-Sec model that maps the attack phases of a given threat to architectural constructs. Figure 5 depicts how this mapping is achieved thanks to a simple extension of the SysML parametric diagram. In particular, each block represents an asset onto which attacks are mapped. Some methodologies aim at defining security

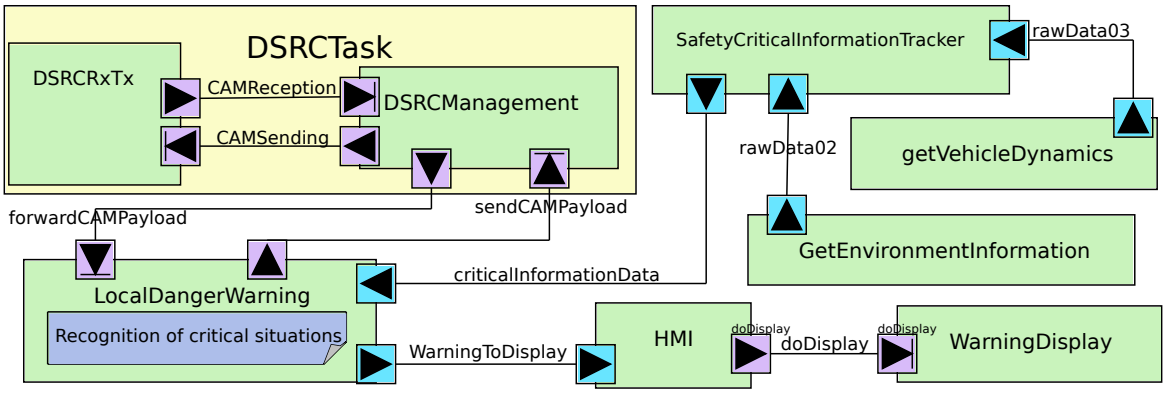

Fig. 4. Information and Event Flow of the EVITA *Local Danger Warning* Use Case (SysML-Sec Block Diagram)

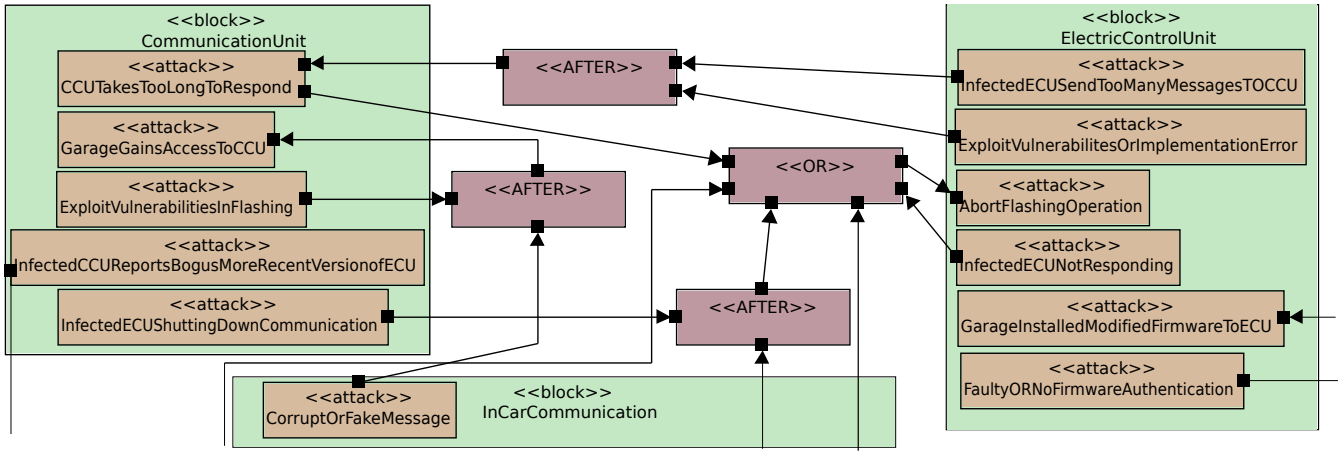

Fig. 5. Attacks Mapped to the Architecture - EVITA *Firmware Update* Use Case (SysML-Sec Parametric Diagram Excerpt)

threats as a fine-grained refinement of the goals of an attacker, like KAOS' antigoals [17]. While it is fine to stepwise refine threats and finally attack steps, we consider that this is not enough to capture all architectural mappings. Plain refinement is satisfactory for attacks that only address computational units and communication channels which are seen as blocks in the SysML parametric diagram. However, the data and event flows featured by embedded systems are distributed over the architecture. Attacks on these assets can be modeled as the combination of attacks on the two previous types of assets: for instance, communication between two ECUs might be threatened by an attacker taking control of an intermediate gateway - e.g., the bridge *CSCU_to_CAN*, see Figure 2 -, then dropping all messages arriving on a channel. We therefore customized SysML parametric diagram constraints to introduce the logical operators like AND and OR, and temporal causality operators like AFTER for representing attack sequences (see Figure 5). Those new constraints are especially helpful to describe the distributed attack scenarios encountered in embedded systems.

D. SysML for Capturing Security Goals

SysML natively supports the notion of requirements. Yet, we have specialized in SysML-Sec the general-purpose SysML requirements by adding a security kind (e.g., privacy, confidentiality, authenticity, integrity, non-repudiation, controlled access, availability, immunity, freshness) to further categorize the security goals and requirements used in our models. The SysML relationships between requirements - e.g., composition, derive requirement and copy - can be applied to security requirements as well as to other requirements of the same kind. For example, in Figure 6, *ControlledAccessToFlashMemory* is composed of two sub requirements: *ControlledAccessToFlashingFunction* and *ControlledAccessToReadFromFlash*. Semantically speaking, R_1 is composed of requirement R_2 and R_3 means that R_1 is more abstract than R_2 and R_3 and that the R_2 and R_3 must be satisfied for R_1 to be satisfied. Integrating security requirements together with other kinds of requirements notably makes it possible to validate evolving requirements through dependency relationships, which can be quite important for safety requirements.

E. Mapping Security Requirements to the Architecture

Even with our extension of the SysML requirement construct with additional semantics, requirements still remain unrelated to assets. As explained before, assets in embedded system are composed of functional and architectural elements of the system, which lead us to map requirements to assets at the partitioning phase. This additionally gives us the possibility

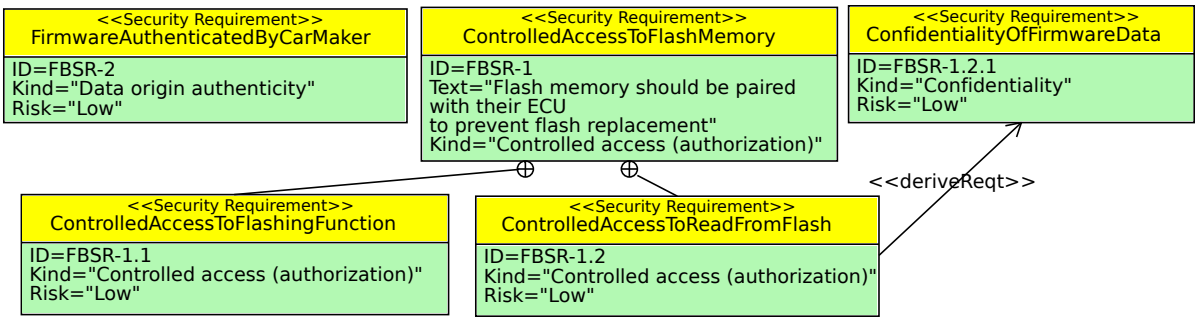

Fig. 6. Security Requirements in the EVITA *Firmware Update* Use Case (SysML-Sec Requirement Diagram Excerpt)

to relate requirements and attacks, since attacks can only be defined from assets.

The introduction of new assets may obviously lead to the definition of new requirements. For example, introducing flash memories at the partitioning phase (see Figure 2) may lead to define requirements dedicated to the security of the software code stored in these flash memories (as pointed, for example, in Figure 6). Introducing flash memories however does not mean that all of them are to store confidential software codes. Those two examples demonstrate two important points: (1) New requirements may be identified when studying candidate architectures at the partitioning phase. (2) Requirements need to be related to architectural elements so as to precise which requirements apply to which architectural elements, and reciprocally.

SysML offers two main facilities to relate instances of SysML constructs together, even if they originate from different views: the *depend* and *allocate* relationships. The first one models an explicit dependence between a master element and a slave element. The allocate relationship is rather used to mention resources that are provided to an element, like for instance a task allocated to a processor in the mapping phase. Finally, relationships between requirements and architectural elements are expressed with a dependency link.

V. Evolving the Architecture Based on Security Requirements

A. From Security Requirements to Security Mechanisms

The security requirements expressed contribute to determine the security mechanisms, like cryptography or access control for instance, that will need to be implemented. However, it is likely that these mechanisms cannot be entirely defined until the end of the architecture specification. We argue that it is still possible and desirable to introduce a description of broad security functionalities into the architecture in order to bridge the gap between the requirements analysis phase and the architecture specification phase. In particular, it is important to describe where code for implementing a security-related algorithm will be run, and how resilient its execution will be. The hypotheses made about implementation resilience when defining requirements should be transferred to the hardware/software partitioning model. By doing so, we also associate a security requirement to an architectural asset, and thus precise which threat its realization may be exposed to: for instance, implementing the access control to the Head Unit of the car with a hardware based implementation will, for remote attackers, be enough to mitigate all attempts at bypassing this control. We illustrate in Section VI to what extent this mapping process can achieve a more complete and accurate threat analyses. Finally, this mapping may even reveal the impossibility of any satisfactory implementation.

Finding which assets are at risk also contributes to the definition of the defense perimeter achieved by a related security objective, and hence to the placement of the security mechanisms that derive from this objective. Typically the possibility of injecting fake traffic from the driver's mobile phone onto the vehicle backbone bus might suggest to filter out mobile phone messages in the engine or chassis and safety domains. This might in turn be implemented through either the authentication of the sender using cryptography or through access control at respective domain gateways based on the origin of traffic.

Due to the communication-centric nature of the threats that we focus on, the introduction of security functionalities will also modify the purely functional information and event flows of the embedded system. We introduce security through the addition of security functions to existing entities, or through the introduction of new entities. Those new entities would typically feature specific security properties that should be matched by the final implementation in the architecture: for instance one such additional entity might reflect the need to use of a tamper-resistant hardware module (and subsequent communication with it) or a gateway defining a trusted domain. Those additional security mappings to the architecture might be introduced in sequence diagrams, or better, in deployment diagrams.

B. Detecting Conflicting Classes of Requirements

Model-Driven Engineering clearly refers to levels of abstractions, and suggests to rely on model transformation techniques to handle the transition between two different levels. The refinement of elements of the system (e.g., splitting a function into two sub functions, splitting a computing hardware domain into two sub domains) is likely to impact the definition of requirements and attacks. Refining requirements

and attacks, or identifying new ones, may also impact the definition of the system design itself. SysML offers the refine operator to express such a refined design, even though no specific refinement methodology is defined.

Security requirements may typically impact other classes of requirements, especially non-functional ones like the interoperability, or more critically for an embedded system, safety. This may result in conflicts. For instance, an excessively large security payload may reduce the available bandwidth of a bus and endanger the safety critical transmission of an emergency braking notification between the brake sensor and the brake actuator. Similarly, security may incur additional costs, like for the verification of the signature of a message, that may entail an an overly long latency for a low-cost ECU. Such conflicts have to be detected.

We consider that, because it focuses on security assets which are architecture-centric, our SysML-Sec methodology is compatible with other requirements engineering methodologies in which the requirements can be linked to the architecture. While we did not experiment with the integration of existing engineering methodologies for other types of requirements, we validated the satisfaction of a few safety requirements through the generation of tests and simulations from our embedded system model [18].

VI. Lessons Learned

Full results of the case study we presented along the paper can be found on the EVITA project web site[1] and in [8]. We extracted around 32 general requirements, i.e., requirements that apply to all use cases. These requirements have been split into four categories: availability (7 requirements), privacy (7 requirements), fake commands (9), and environment-related requirements (9). General requirements have been progressively extracted from requirements specific to the first analyzed use cases. They could probably constitute a good foundation for defining security-oriented patterns. Then, for each use case, we have defined specific requirements. For example, for the *firmware update* use case, 9 additional security requirements have been elicited: some of them are displayed in Figure 6. Both general and specific requirements are linked to assets and attacks. The iteration between requirements, architecture, and attacks also led us to identify much more attacks than when simply identifying them based on our security expertise and on brainstorming sessions: the number of identified attacks was multiplied by a factor between 2 and 4, depending on use cases.

Achievements. One of our initial objectives was to achieve a comprehensive and more systematic analysis of security issues of an automotive on-board system. Our methodology has significantly improved the elicitation of security requirements. For instance, the adherence to a stepwise and iterative process helped us a lot to resist the constant urge to jump directly to the selection of particular security mechanisms for the system design. This would have likely resulted in a less complete insight into the threats to the vehicle, and in a poorer understanding of the dependencies between security requirements. Similarly, the use of the architecture and function placement helped us to understand the use cases, as well as to point at some inconsistencies regarding security requirements they expressed. For instance, there was no need to ask for the non repudiation of a firmware update, while users were specifically asking for the use of a digital signature in their use case.

The result of our analysis can also be compared with the work of Bar-El [19] on automotive on-board networks. The latter work captures a number of security requirements only focusing on the cryptographic protocols to be deployed in an onboard network. However, [19] fails to capture hardware/software partitioning or the compatibility of security mechanisms with safety requirements, that are so important in vehicles, like for instance the delay required to send and verify a message. In comparison, we relied on the mapping of security requirements with assets and with the information flows to later evaluate the satisfaction of safety, realtime, or performance requirements through formal validation [18], as well as through tests and simulations in the same open-source toolbox [20]. We furthermore contend that our analysis has also captured higher level security requirements as it starts with security properties rather than mechanisms and describes their dependencies.

Limitations. Our methodology does not bridge the implementation gap. In particular, we do not describe requirements as to the effectiveness of the implementation of the security mechanisms that realize the requirements described. Most notably, secure programming issues, like buffer overflows, are plaguing all kinds of software produced, yet they are not the result of a poor design of the architecture as we can describe it in SysML and UML. Similarly, hardware based implementations can be poorly done, and for instance be vulnerable to timing or power attacks. We believe that addressing such issues is not an architecture design issue per se, and can be achieved by the systematic application of software or hardware security hardening engineering recipes. Nor our methodology nor the tools we developed can provide support for these tasks (though they might document such weaknesses for COTS).

VII. Related Work

There has been a quite remarkable progress in the area of security requirements engineering in the past decade. In [21], Nhlabatsi et al. classify security requirements engineering work in software systems according to four dimensions, namely: (1) *goal-based approaches*, (2) *model-based approaches*, (3) *problem-oriented approaches*, (4) *process-oriented approaches*.

Our own proposal essentially associates a goal-oriented description of security objectives (loosely inspired by the KAOS framework [17]), with a model-driven approach to

[1]TTool models are available at the following link: http://www.evita-project.org/Deliverables/evita_t2300_23.xml

system design and in particular to the refinement of security assets.

In particular, we benefit from the refinement and tracing qualities of goal-based approaches that have been similarly successful in other domains of requirement engineering. Another strength of goal-oriented approaches lies in their ability to capture dependencies even between security requirements with a high-level of abstraction.

We emphasize the importance of assets for setting goal-based requirements, in accordance with the views expressed in [22], [23]. In contrast however, we consider that an embedded system features assets of a very different nature at different levels of the architecture which we do not intend to interconnect in a complete causal graph (though we have introduced a few limited and local relationships on which we perform consistency checks).

Models on the other hand are extremely good at capturing architecture details and have been shown an excellent fit for describing security requirements regarding cryptographic protocols. For instance, methodologies like UMLSec [24] focus on the mapping of security mechanisms to the software architecture. To our knowledge, no such approach has however addressed hardware and co-design issues. In our framework, security requirements and functions can be progressively refined until a formal verification step integrated in our toolkit [15]. TTool implements model transformation techniques in order to translate refined and formally expressed security requirements and designs into a pi-calculus specification taken as input by ProVerif, a Dolev-Yao based security proof toolkit [25].

As mentioned before, our approach aims at achieving a viable design. Other proposals have also hinted at the need to determine the right tradeoff satisfying security requirements as well as functional or other non-functional requirements [26], [27], [28], [29]. Our methodology that introduces security requirements into the SysML framework further enriches these contributions through the determination of quantitative evaluation of the satisfaction of this tradeoff with the support of model-driven tools. For instance, we simulated the CAN bus to evaluate the impact of a cryptographic envelope implementing our authentication requirements [20]. We also evaluated the braking latency in the EVITA emergency braking use case by automatically generating software for a virtual prototyping environment from AVATAR models [30]. Some of these validations can be described even at the goal level description of security requirements through the use of SysML testcases.

It is worth mentioning that the model-driven engineering of requirements has long been supported by researchers in the field of embedded systems [31], [32], [33]. However, only Peraldi et al. [34] advocate the need to link the model driven engineering of the system architecture and a goal-oriented expression of requirements that we follow in our approach. To our knowledge, none of these proposals has addressed the expression of security requirements.

Our methodological proposals also share quite some similarities with the TwinPeaks approach advocated by Nuseibeh [35], although the latter only deals with software systems. Instead of a simple spiral alternating between the requirements and the architecture as TwinPeaks suggests, we alternate between the Y-Chart modelling of software and its mapping to hardware components, the identification of assets and threats to them, and the identification of security requirements. In particular, we also deal with the three management concerns that TwinPeaks aims at addressing: (1) exploring the solution space (in our case, both the embedded system architecture and attacks that may result out of this design) early makes it possible to incrementally provide feedback about requirements; (2) the designer has to rely on commercial off-the-shelf software (as for TwinPeaks), or available electronic components, or standard cryptographic algorithms and requirements (security requirements in our proposal) help narrow down their proper selection; (3) rapid change, which is also very much linked with refining the architecture in our case.

VIII. Conclusion

We have introduced SysML-Sec, an environment to support the security requirement engineering process in an embedded system and an open-source tool (TTool) to support this process. We conducted an experiment on an automotive on-board system under definition, which helped us assess the adequacy of our approach. Out of this experience, our main claim is that a security requirements engineering methodology should capture the relationship between security concerns and the system architecture. Security requirements should be mapped onto this architecture and in turn, influence the system architecture through an iterative refinement design process extending the classical Y-chart approach used in embedded systems.

We also believe that our adoption of a SysML supported model-driven refinement of security assets might be used right from the early definition of security requirements to link the security expert's goal-oriented point of view with the model-centric perspective of the embedded system designer and enable their collaboration before any of them commits to an inappropriate solution.

We plan to further investigate the question of the validation of requirements. We have already experimented with assessing how safety is possibly impacted by the security mechanisms introduced after security requirements, like for instance, assessing the added latency for performing a braking operation with secure communication. We also plan to introduce security-oriented reasoning capabilities into our modeling environment.

References

[1] A. Huang, "Keeping Secrets in Hardware: the Microsoft XBox Case Study, AI Memo 2002-008, Massachusetts Institute of Technology, Artificial Intelligence Laboratory," Tech. Rep.

[2] F. Assolini, "The Tale of One Thousand and One DSL Modems, kaspersky lab," Oct. 2012.

[3] S. Esser, "iOS Kernel Exploitation," in *BlackHat 2011*, 2011.

[4] A. Costin and A. Francillon, "Ghost in the Air(Traffic): On insecurity of ADS-B protocol and practical attacks on ADS-B devices," in *BLACKHAT 2012, July 21-26*, Las Vegas, USA.

[5] H. Teso, "Aircraft Hacking," in *HITB Security Conference*, Amsterdam, The Netherlands, 2013.

[6] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, and S. Savage, "Experimental Security Analysis of a Modern Automobile," *In Proc. of the 31st IEEE Symposium on Security and Privacy*, May 2010.

[7] O. Henniger, L. Apvrille, A. Fuchs, Y. Roudier, A. Ruddle, and B. Weyl, "Security Requirements for Automotive On-Board Networks," in *Proceedings of the 9th International Conference on Intelligent Transport System Telecommunications (ITST 2009), Lille, France*.

[8] A. Ruddle and et al, "Security Requirements for Automotive On-board Networks Based on Dark-side Scenarios," EVITA Project, Tech. Rep. Deliverable D2.3, 2009.

[9] OMG, "SPT: Profile for Scheduling, Performance and Time," 2005.

[10] J. Vidal, F. de Lamotte, G. Gogniat, P. Soulard, and J.-P. Diguet, "A Co-Design Approach for Embedded System Modeling and Code Generation with UML and MARTE," in *Design, Automation and Test in Europe Conference and Exhibition (DATE)*, April 2009, pp. 226–231.

[11] OMG, "SysML: Systems Modeling Language," 2012.

[12] F. Balarin, Y. Watanabe, H. Hsieh, L. Lavagno, C. Passerone, and A. Sangiovanni-Vincentelli, "Metropolis: An Integrated Electronic System Design Environment," *Computer*, vol. 36, no. 4, pp. 45–52, April 2003.

[13] L. Apvrille, "TTool website," in *http://ttool.telecom-paristech.fr/*, 2013.

[14] L. Apvrille, W. Muhammad, R. Ameur-Boulifa, S. Coudert, and R. Pacalet, "A UML-based Environment for System Design Space Exploration," *13th IEEE International Conference on Electronics, Circuits and Systems, 2006*, pp. 1272–1275.

[15] G. Pedroza, D. Knorreck, and L. Apvrille, "AVATAR: A SysML Environment for the Formal Verification of Safety and Security Properties," in *The 11th IEEE Conference on Distributed Systems and New Technologies*, Paris, France, May 2011.

[16] C. Haley, R. Laney, J. Moffett, and B. Nuseibeh, "Security Requirements Engineering: A Framework for Representation and Analysis," *Software Engineering, IEEE Transactions on*, vol. 34, no. 1, pp. 133 –153, Jan.-Feb. 2008.

[17] A. Van Lamsweerde, "Engineering Requirements for System Reliability and Security," *Software System Reliability and Security*, vol. 9, pp. 196–238, 2007.

[18] G. Pedroza, M. S. Idrees, L. Apvrille, and Y. Roudier, "A Formal Methodology Applied to Secure Over-The-Air Automotive Applications," in *VTC-Fall2011, IEEE 74th Vehicular Technology Conference, San Francisco, USA, 5-8 September 2011*.

[19] H. Bar-El, "Intra-Vehicle Information Security Framework," *Design*, pp. 1–19, 2009.

[20] B. Weyl, M. Wolf, F. Zweers, T. Gendrullis, M. S. Idrees, Y. Roudier, H. Schweppe, H. Platzdasch, R. E. Khayari, O. Henniger, D. Scheuermann, A. Fuchsa, L. Apvrille, G. Pedroza, H. Seudie, J. Shokrollahi, and A. Keil, "Secure On-board Architecture Specification," EVITA Project, Tech. Rep. D3.2, 2010.

[21] A. Nhlabatsi, B. Nuseibeh, and Y. Yu, "Security Requirements Engineering for Evolving Software Systems: a survey," The Open University, Tech. Rep. 1, 2010.

[22] M. Salehie, L. Pasquale, I. Omoronyia, R. Ali, and B. Nuseibeh, "Requirements-driven adaptive security: Protecting variable assets at runtime," in *Requirements Engineering Conference (RE), 2012 20th IEEE International*, 2012, pp. 111–120.

[23] L. Pasquale, M. Salehie, R. Ali, I. Omoronyia, and B. Nuseibeh, "On the role of primary and secondary assets in adaptive security: An application in smart grids," in *Software Engineering for Adaptive and Self-Managing Systems (SEAMS), 2012 ICSE Workshop on*, 2012, pp. 165–170.

[24] J. Jürjens, "UMLsec: Extending UML for Secure Systems Development," *5th International Conference on the Unified Modeling Language*, pp. 412–425, 2002.

[25] B. Blanchet, "Automatic Verification of Correspondences for Security Protocols," *Journal of Computer Security*, vol. 17, no. 4, pp. 363–434, Jul. 2009.

[26] G. Elahi and E. S. K. Yu, "A goal oriented approach for modeling and analyzing security trade-offs," in *Proceedings of the 26th International Conference on Conceptual Modeling*, 2007.

[27] S. Houmb, G. Georg, J. Jürjens, and R. France, "An integrated security verification and security solution design trade-off analysis approach," pp. 190–219, 2007.

[28] S. Lee, "Probabilistic risk assessment for security requirements: A preliminary study," in *Proceedings of the 5th International Conference on Secure Software Integration and Reliability Improvement*, 2011, pp. 11–20.

[29] Y. Asnar, P. Giorgini, and J. Mylopoulos, "Goal-driven risk assessment in requirements engineering," in *Proceedings of RE'2011, vol. 16, no. 2*, 2011, pp. 101–116.

[30] L. Apvrille and A. Becoulet, "Prototyping an Embedded Automotive System from its UML/SysML Models," in *ERTSS'2012*, Toulouse, France, Feb. 2012.

[31] M. Broy, "Requirements Engineering for Embedded Systems," in *Proceedings of the Workshop on Formal Design of Safety Critical Embedded Systems (FemSys'97)*, Apr.

[32] M. von der Beeck, P. Braun, M. Rappl, and C. Schröder, "Model Based Requirements Engineering for Embedded Software," *2012 20th IEEE International Requirements Engineering Conference (RE)*, vol. 0, p. 92, 2002.

[33] E. Geisberger, J. Grunbauer, and B. Schatz, "Interdisciplinary Requirements Analysis Using the Model-Based RM Tool AUTORAID," *Automotive Requirements Engineering, International Workshop*, vol. 0, p. 1, 2006.

[34] M.-A. Peraldi-Frati and A. Albinet, "Requirement Traceability in Safety Critical Systems," in *EDCC2010 - Workshop on Critical Automotive applications: Robustness and Safety (CARS'2010)*, ser. ACM International Conference Proceeding Series, J.-C. Fabre, O. Guetta, and M. Trapp, Eds. Valencia, Espagne: ACM, Apr. 2010, pp. 11–14.

[35] B. Nuseibeh, "Weaving Together Requirements and Architectures," *IEEE Computer*, vol. 34, no. 3, pp. 115–117, 2001.