A Framework Towards the Efficient Identification and Modeling of Security Requirements

Muhammad Sabir Idrees (idrees@eurecom.fr)*
Yves Roudier (yves.roudier@eurecom.fr)*
Ludovic Apvrille (ludovic.apvrille@telecom-paristech.fr) †

Abstract: Security concerns in vehicular embedded systems have made requirement engineering one of the most critical phases when designing those systems. This paper introduces a new framework that follows a Model Driven Engineering (MDE) approach and targets the identification and modeling of security requirements at early design stages.
In particular, the system specification is provided through use cases. From that description, functional, architectural, and mapping views are constructed, using UML diagrams. Based on both use case specifications and system views, possible attacks and security requirements are identified and modeled using SysML diagrams. These identified security requirements serve as the basis for a trustworthy communication among different entities, and can be further used and refined in next methodological stages. The overall methodology is already implemented in a toolkit - called TTool - and is exemplified in the context of a vehicular-based application studied in the EVITA European project.

Keywords: Security Requirements Engineering, Attack Modeling, Requirement Modeling.

1 Introduction

The increasing complexity of large-scale heterogeneous systems such as embedded systems has made requirements engineering one of the most critical phase during system conceptualization. Security requirements in particular are more and more are stake with the development of networked embedded systems: PCs are no more the only computer systems targeted by attackers. However, determining security requirements within embedded systems generally and paradoxically necessitates a detailed enough knowledge of the system components and interactions, like how functions are mapped onto hardware, whether some given communication might be seen by an attacker, or not, etc.
Security requirements probably constitute one of the most abstract documentation of the expected system behavior. These requirements should provide a specification that has to be satisfied at every subsequent stage of the system: analysis, design, implementation, and validation/testing. Establishing relationships between requirements and such later phases of engineering should thus receive appropriate support: for instance, it should be possible to document the fact that some security mechanism is introduced in order to satisfy one

* EURECOM, Sophia Antipolis, France
† Institut Telecom, Telecom ParisTech, CNRS LTCI, Sophia Antipolis, France

security requirement, or to point at some test over the implementation in order to verify that it is compliant with the same requirement. Additionally, in the case of embedded systems, the need for hardware protection to satisfy security requirements should be supported by the methodology used.
Security requirements should furthermore constitute a manageable documentation for the average system engineer. As of today, requirements are mainly defined using natural language descriptions [PGH06] and are largely text based. However, such techniques are often imprecise and may lead to the specification of inconsistent security requirements. In particular, security requirements are often defined independently from the security threat analysis or on the contrary, they are mixed together. A precise description of a separately defined threat coverage is however necessary to provide convincing arguments as to the security achieved by the system under design. Furthermore, a text-based description also does not make it possible to define relationships between different requirements to organize the set of security requirements into a description with different levels of complexity that would be more manageable by a human being. In that respect, graphical formalisms typically exhibit many advantages in terms of human readability, although text-based tables may be more appropriate for checking the consistency of a set of requirements.
Although there has been several approaches to identify, define, and refine security requirements, so far no approach integrates in one single environment threat identification, security requirements modeling, refinement, formal verification, code generation, and test cases. In addition to the identification and modeling of security requirements, the major innovation in the framework we introduce is the integration of threat modeling, semi-formal security requirements, semi-automated refinement, the Y-chart approach to hardware/software co-design [LvdDV99], formal verification, and code generation, all into one environment (TTool) [TTo]. Moreover, our methodology follows a Model Driven Engineering (MDE) approach.

This paper provides a framework for security requirements engineering for vehicular on-board networks. Section II reviews the capabilities of the existing approaches in terms of security requirement identification and modeling. The following section defines the framework for security requirements in the context of vehicular embedded systems. In section IV, we evaluate our framework for deriving security requirements in the context of the EVITA project which features use cases [KFM+09] that require securing an automotive on-board system. Finally, a conclusion summarizes the results that were attained from the security requirement framework and presents future work.

2 Related Work

2.1 General Security Requirements Approaches

There has been studies for modeling, specification and analysis of application-specific security requirements [vL04]. The extended framework presented in their approach, addresses model (system goals) of the system to be covered and emphasizes anti-goal model set up by attackers. An anti-model elicited from goal model exhibits how security approaches could be maliciously threatened by attacker. Security concerns are captured and refined until reaching security requirements. In [GSWY04][MI05][MGI05], a Security Quality Requirements Engineering (SQUARE) methodology for eliciting and prioritizing security

requirements is presented. The SQUARE methodology is composed of nine steps, in order to provide a means for electing, categorizing, and prioritizing security requirements for information technology systems and related applications. However, in SQUARE, it is not specified how security requirements can be modeled, whereas, most of the existing approaches are defined using text-based description: this makes it difficult to organize the set of security requirements into a description with different levels of complexity. Another disadvantage is the lack of support for requirement traceability making that approach not suitable for complex systems.
Many specification approaches [HF04][WJY09][Jür02][LBD02] have followed the road of defining security requirements based on the UML profiles [UML]. In addition to the graphical specification, portability and interoperability are usually the strength of such profiles. A interesting feature of KAOS [HF04] is that goals and related constraints can be formally defined using temporal logic. However, in order to define such expressions accurately, design should be as precise as possible. Another drawback of KAOS is its systematic coverage of threats, and also it does not provide any support for software-hardware co-design nor code generation and testing capabilities. In [Jür02], UMLSec introduces a security-oriented methodology based on activity diagrams, state-charts, sequence diagrams, class diagrams, and deployment diagrams. The UMLsec methodology is more about security mechanisms design than about security requirements, linked with use cases. SecureUML [LBD02] is a profile that aims to provide security modeling capabilities. SecureUML however only specifies security polices for Role-Based Access Control (RBAC) using graphical notation and logical constraints, so it is essentially adapted to application security.

2.2 Security Requirements for Vehicular Communication System

Recent work outline challenges for securing inter vehicular communication. Attacks and security requirements specific to Car-to-Car (C2C) and Car- to-Infrastructure (C2I) are described in [RH07][ABD+06][KMS06]. In [ABD+06], a model of attacks on an inter-vehicle communication system is introduced. [Sch99] introduces an approach to capture the attack trees and subtrees. These attack trees correspond to specific attacks on C2C, C2I, C2Home applications. Based on identified attacks, several security requirements are specified. However, in this approach, requirements are strongly related to security mechanisms. But security requirements rather not make any assumption regarding possible realizations. Indeed, as explained in [HLMN08], it is common to express security requirements describing security mechanisms to be used. Security requirements are defined in terms of *what* and *why* objects need protection and also describing *how* objects are to be protected. For example, the Common Criteria - Protection Profile (PP) [CC] identifies security requirements for a class of security devices for a particular purpose, such as protecting the access to network devices by applying network firewalls. In our framework, security requirements are constraints arising from security concerns; thus these requirement do not specify how the constraints are satisfied, therefore leaving the *how* to the decision of security experts who shall select the most secure and appropriate solution, at the time.
Another attempt to find security requirements in Vehicular Ad-Hoc Network (VANET) application is presented in [KMS06]. In their requirement engineering process, cluster analysis approach is applied for grouping similar security requirements into respective categories. The objective of cluster analysis is to group similar applications and only

considering one representative of the group. In other words, this cluster analysis aims at sorting different VANET applications in a way that the degree of association among different application is maximum, if they belong to similar security property group. Then, a small subset of respective applications from each cluster is selected and analyzed in more detail. Potential attacks are defined against each application. Furthermore, security requirements are derived for each application that will prevent these attacks. However, in this techniques security requirements are also defined using text based description.
An analysis of existing work shows that security requirements are mostly defined using text-based description: security engineers have to intervene to provide the information at later stages of system development for specifying requirement relationships and traceability. Requirement traceability that can help engineers to determine the origin of the security requirement, but existing work may not satisfy these properties. In contrast, our security requirement framework yields a semi-formal definition of security requirements, graphical security requirement modeling environment, defines requirement dependencies and requirement traceability properties.

3 A Framework for Security Requirements Engineering

3.1 Automotive Environment: An Example of a Complex System

Developing a model of the system to be analyzed usually requires that this model already exists or that this system is fully specified. In the context of the envisaged vehicular embedded system (EVITA project [EVI]) things are slightly different because the purpose is to contribute to the design of the security architecture design and of the protocol by specifying security requirements. Thus, our framework shall be applicable to a system that is neither available nor fully specified. A reference architecture [KFM+09], as shown in figure 1, is defined for illustrating the internal architecture of the vehicular on-board network. This on-board network contains more than 70 Electronic Control Units (ECUs), electronic sensors and actuators interconnected by buses and organized in different domains. On-board networks are assumed to operate in an uncontrolled environment. Therefore, their assets must be protected against a wide variety of attacks.

Analyzing the design of an embedded system network requires system engineering processes to document system-wide security requirements intensively and to trace those security requirements at different levels of system abstraction. The most relevant factor in EVITA is to determine security requirements without considering security mechanisms. To make those security requirements robust, relationships between security requirements and particular respective dependencies among security requirement is also required. In our framework, security requirements are described with an approach close to the language of system engineers [IEEb][IEEa]. The primary goal is to offer a framework dedicated to security requirements, flexible enough to be adapted to any actual system but still capable of producing accurate security requirements. The security requirement framework is composed of three main elements: *Specification*, *Security requirement engineering process* and *Requirement refinement* (see Figure 2). However, in the rest of the paper we are only focusing on **Security requirement engineering process**.

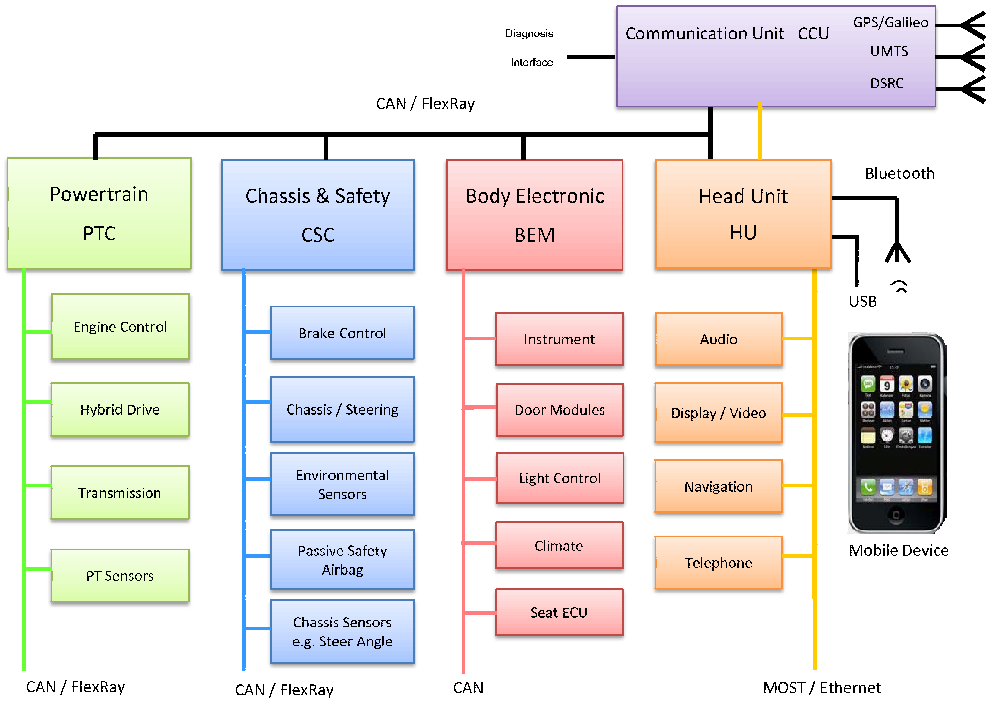

Fig. 1: EVITA On-board Reference Architecture

3.2 Security Requirement Engineering

3.2.1 Threat identification and Modelling

An essential part of the *security requirement engineering process* (see Figure 2) is the threat identification and the threat modeling elements. We are considering two kinds of attacks on vehicular embedded system, based on the functionalities identified in the use cases [KFM+09]: *Computing attacks* and *Communication attacks*. Computing attacks consist of physical modification of the component such as modifying the content of an embedded ROM, whereas communication attacks are targeting communication link. For instance, listening, injecting, and modifying communication in vehicular network.
Existing work on threat modeling [SS02] [ABD+06] [PS] are focused on an attack tree approach [Sch99]. If attack trees are well defined structures, there is no work on automatic or semi-automatic linking of attack trees with other UML modelling diagrams [UML]. In our proposed framework, SysML Parametric Diagrams (PD) [Sys] are used to model asset-based attacks. Using SysML PD to model attack trees makes it possible to link attacks to SysML Requirements Diagram (RD) [Sys], and to other UML/SysML elements, therefore facilitating the attack coverage analysis. More precisely, in SysML PD, asset-based attacks are defined through PD graphical nodes that represent *constraints* or *value types*: constraints are meant to relate value types together. As shown in Figure 3, value types can represent attacks, while constraints are meant to model relations between attacks. To describe the relationship, dependencies, and timing constraint between threats, a new set

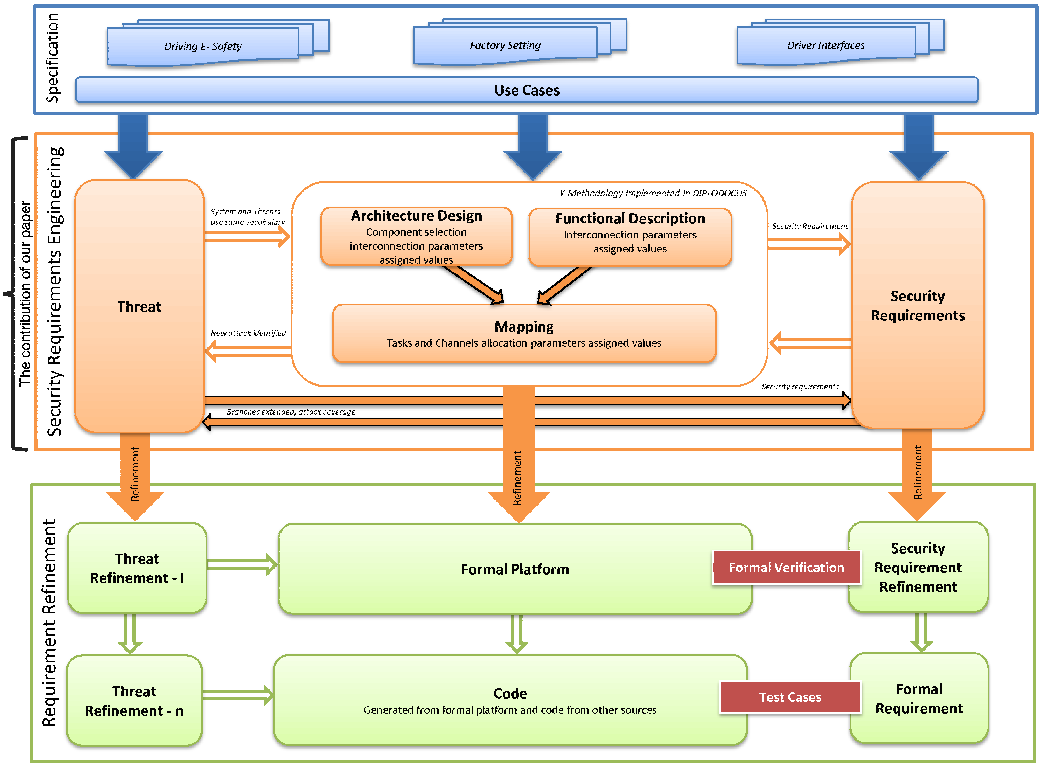

Fig. 2: Security Requirements Design Methodology

of parametric constraint are defined: *or*, *and*, *sequence*, *after*, and *before*. For instance, *before* and *after* are defined to model temporal constraints between attacks. Additionally, SysML *block* and *internal block* represent the main assets of an on-board network: ECUs, sensors, actuators, communication links and safety critical applications running on in-vehicle devices. Attacks on these assets depict the ways an attacker could pursue his/her goals.

3.2.2 Integration of the Y-Methodology

Another characteristic of our security requirement framework is the integration of the Y-methodology [LvdDV99] approach to identify security requirements. In our framework, the DIPLODOCUS Y-like approach [AMAB+06] is used. It fits nicely with our system architecture description, where we need to define potential security requirements for targeted functions expected to run both in software (applications running on CPUs) and hardware (hardware accelerators, sensors, actuators). In [LvdDV99], a modeling and simulation methodology that allows for efficient architecture exploration of heterogeneous embedded media systems is explored. A clear distinction between application and architecture is made to explore design space. Multiple targeted applications are mapped onto candidate architecture in order to provide the reconfiguration property for an embedded system. This approach also facilitates to decide possible architecture design for the time critical

application tasks running onto different hardware. For instance, which application tasks are mapped onto which system components. This is also very important and critical for vehicular embedded systems, where many safety critical and non-safety critical applications run on in-vehicle devices.

3.2.3 Identification and Modelling of Security Requirements

The aim of specifying security requirement is to provide the input to the secure on-board architecture design to the model-based verification of on-board architecture and protocol specification, to the security architecture implementation. The *functional path and mapping* approach is defined in order to identify security requirements. An iterative process is employed, consisting of the following steps:

1. Extract security requirements from use cases and thread model
    - Derive functional view
    - Derive architectural mapping and correct functional view accordingly, if necessary
2. Verify coverage of security requirements
    - Attack coverage
    - Use case consistency/completeness
3. New identified attacks
    - Generate new security requirements for unmatched threats or change use cases
        - Re-evaluate threat coverage

Additional benefits of this approach are a more precise definition of the use cases, the verification of existing attacks, the identification of new attacks, and a more explicit mapping of security requirements to functions and assets.
In our proposed framework, security requirements are modeled in SysML Requirement Diagrams (RD). In [Bal06], capabilities of SysML diagrams are explored to describe the different aspects of an automotive embedded system such as context, requirement, and behavior of the system. [Bal06] deployed different SysML profiles to support the conceptual stages of the product development life-cycle. For instance, decomposition of system needs and representing them as requirements in the model. The SysML Block Definition Diagram (BDD) is used to manipulate the structure of the system, whereas the system behavior is specified by using *Interaction*, *State Machine* and *Activity* diagrams. Another characteristic of a SysML is: *Requirement Containment* and *Derive Dependency* formalisms used to define relationships between requirements. We have more particularly used SysML *containment, dependency - deriveReqt*, and reuse in different namespaces *copy* relationships for defining security requirements. As explained in [Sys], the *containment* relationship can contain multiple sub-requirements in terms of hierarchy and enables a complex requirement to be decomposed into its containing child requirements whereas, *deriveReqt* determines the multiple derived requirements that support a source requirement. These requirements normally present the next level of requirement hierarchy. A

Security Requirement stereotype is introduced to make clear distinction between functional requirements and security requirements of the system. In this way system engineers can model both functional and non-functional requirements of the system in one modeling environment. Furthermore, a *Kind* parameter is defined to specify the category of the security requirement such as, *confidentiality*, *access control*, *integrity*, *freshness*, etc,.

3.3 Security Requirements Modelling Environment

We have applied the aforementioned methodology using TTool [TTo]. TTool[1] is an open-source toolkit initially developed for the TURTLE UML profile [ACLdSS04]. TTool includes diagramming facilities, formal code generators and graph analysis tools. It now supports several other UML profiles such as the *DIPLODOCUS* profile [AMAB+06]. DIPLODOCUS includes both functional, architecture and mapping views (Y-Methodology), SysML RD, and SysML PD, which have been used for the related methodological steps.

4 Applying the Security Requirement Framework in the context of Vehicular Embedded System

By manipulating the security requirement framework one can identify the threats, security requirements and classify the security requirements determined in addition to their security properties. One or several typical use cases [KFM+09] are defined in EVITA for communication with on-board embedded system. Use cases are defined in terms of the communication entities (e.g. vehicles, driver, backend infrastructure), communication relations, and describe the technical details as well as exchanged information.

The considered use case defines the process of OBD firmware flashing process. To start the diagnosis session, the car has to be activated. The ECU initializes its software and starts the diagnosis function, called diagnosis server. A diagnosis request is then sent via the Communication Unit (CCU) (on-board diagnosis interface) to the ECU. The ECU authenticates the diagnosis tool and checks the data integrity. If the request is successful, the ECU opens a programming session. The diagnosis tool then sends the encrypted packets of the new firmware to the ECU, which stores it in the RAM. The new firmware is decrypted at ECU level and flashed in the ROM packet wise. This use case will serve as a basis for the deduction of security requirements, such as authentication, integrity. However, these basic requirements are very generic. These requirements are also required as to identify the potential for malicious attacks and to specify requirements in depth.

4.1 Attacking the Firmware Flashing Process

There are two possible ways to attack vehicular on-board networks through the firmware flashing process: *Active attacks* modifying the behavior of the system and *Passive attacks* aiming at information retrieval without modifying the behavior. An example scenario of attack consist of an attacker abusing the flashing process itself in a service station and installing modified firmware into on-board ECU. On the vehicle side, even if all security checks (authenticity, integrity, confidentiality, etc.) are performed, the attacker can get access to other domains of the on-board network. The attacker can also get access to the CCU by exploiting vulnerabilities in protocol implementation. If it is not the case,

[1] **T**URTLE **Tool**kit

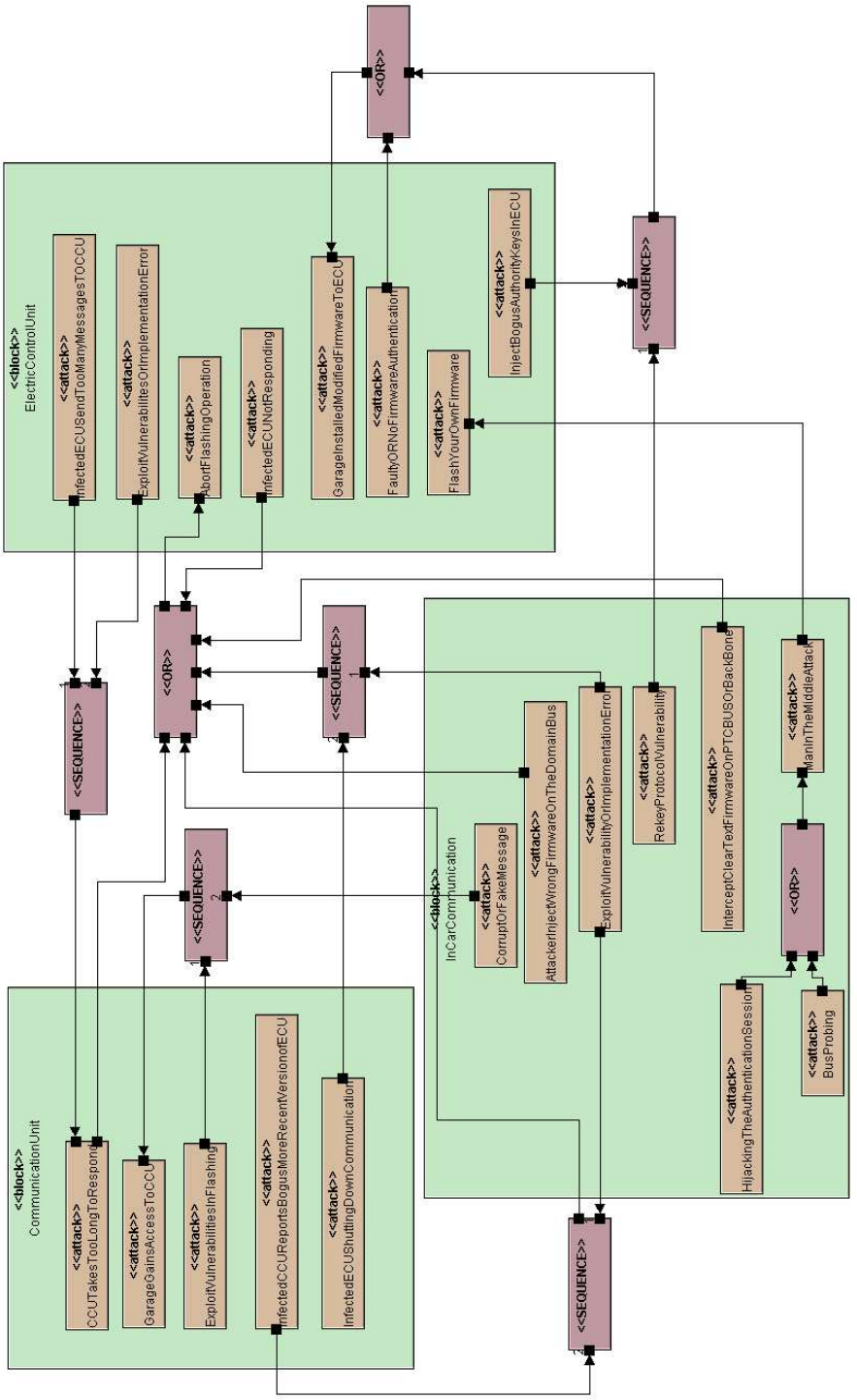

Fig. 3: Attacks on Firmware Flashing process

it is still possible to attack the CCU via the Head Unit (HU) through internet access or by personalizing the car with external devices such as a PDA, a Laptop, etc, to abort the firmware flashing process. Another possible attack could consist in injecting bogus authority keys into ECUs, which allows the attacker to send fake messages to other domains within the on-board network or broadcasting the fake warning messages to other vehicles and Roadside Units (RSU). More detailed asset based attacks on vehicle on-board network, during firmware flashing process, are shown in Figure 3.

4.2 Functional and Mapping View of the Firmware Flashing Process

The functional description of use cases is modeled as a graph of communicating tasks. The hardware architecture is built by selecting, connecting and characterizing generic components such as, CPUs, I/O devices, hardware accelerators, storage nodes (RAM etc). Then, communicating tasks of the function description are mapped onto architecture components: each task is allocated to a computing node, each logical communication channel is allocated to physical links and memories. Additionally, mapping parameters are set, e.g. arbitration policies for buses, priorities of tasks mapped on CPUs.

A functional specification has been made and then mapped on the reference architecture as shown in Figure 4. For example, *DiagnosisRequestManagement* function is mapped on the *CCU* CPU. As a result of applying the Y-methodology, a detail set of system-oriented security requirements can be specified in a finer grained manner, in particular the functional path of use cases can be clearly identified. For example, the path between *DiagnosisRequestManagement* and *firmwareIdentification* takes four different buses.

4.3 Identified Security Requirements

An analysis of attacks, functional and mapping view shows that specific attacks may contribute to different attack objectives and to different attacks. Several security requirements are defined to perform secure firmware flashing and to protect the system against general attacks. Due to lack of space, we highlight the most relevant and important security requirements identified for the firmware flashing process and present one instance of SysML RD (see Figure 5). However, the detailed SysML RD models are presented in [RWW+09].

- *Controlled Access to Flash Memory*: This set of security requirements ensures that a flashing update takes place with authorized firmware, whose IPR is not endangered. Whenever a flashing process is performed, a controlled access to flash memory requirement may be specified. This will ensure that flash memory should be paired with their ECU to prevent flash replacement. Controlled access to flash memory requirement has a containment relationship which contains; *Controlled access to flashing function* and *controlled access to read from flash* requirements.

- *Ensure Availability of ECU/CCU*: This set of security requirements are focusing on properties that should be maintained despite denial of service attacks, coming either under the form of computational resource oriented DoS, network DoS, or even degradation of real-time constraints. An availability property applies to a service or to a physical device such as CPU, RAM or Bus, as shown in Figure 5.

- *Prevent Man in the Middle Attack*: This security requirement prevents attackers from performing Man in the Middle attack. Using *deriveReqt* relationship further

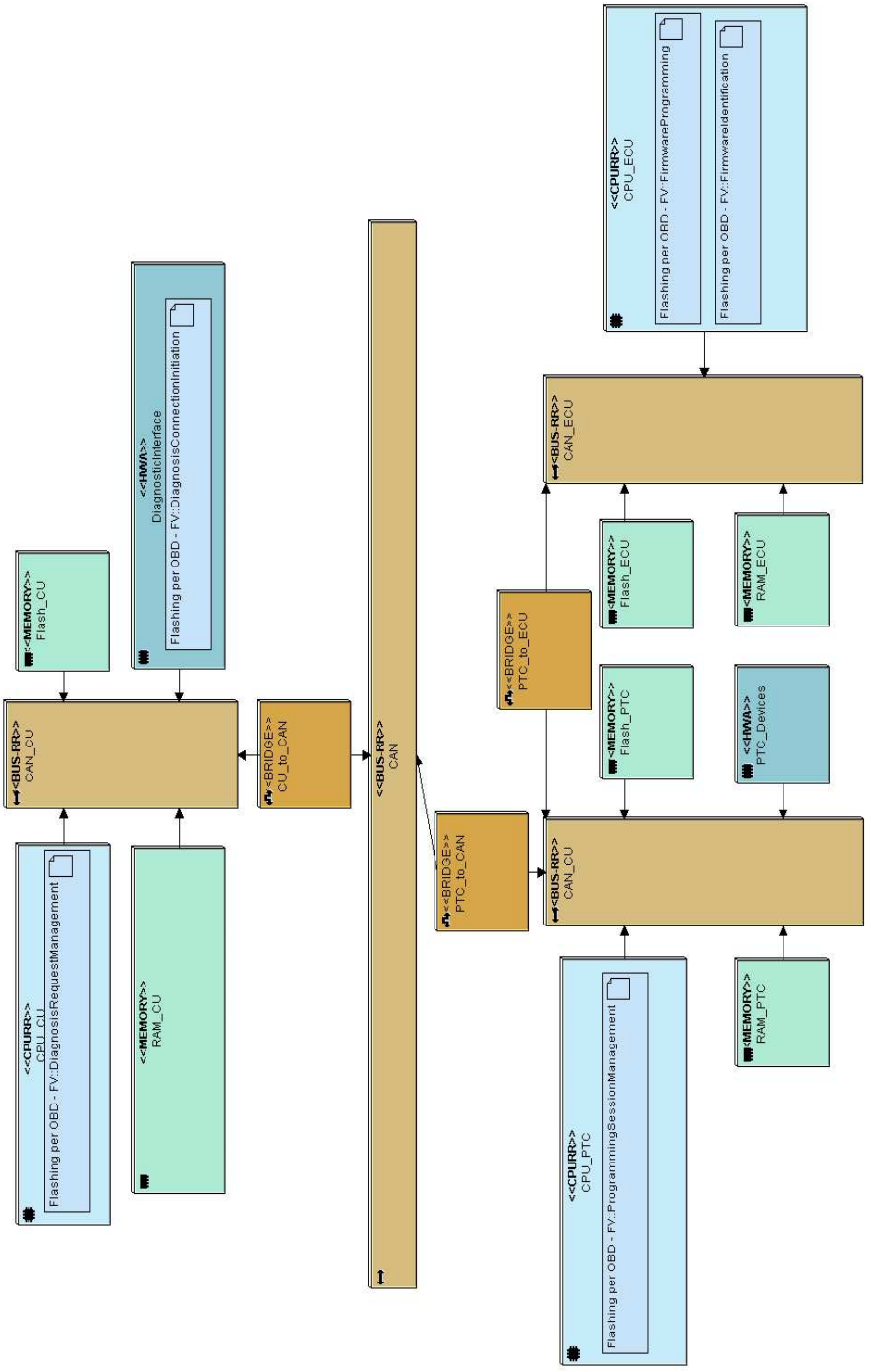

Fig. 4: Firmware Flashing Mapping View

security requirements, integrity of message attributes along functional path, authentication of functional path and message freshness along functional path requirements are derived.

- *Protect ECU Flashing Process* : This security requirement prevents attackers from sending wrong or fake commands from within or from outside the TOE to stop flashing process. Using deriveReqt and copy relationship further security requirements, Flashing Command Freshness and Code Origin Authenticity requirements are derived from this requirement. whereas, code integrity is defined using copy relationship.

- *Secure Approach for Software Development*: Formal verification and software vulnerability assessment is required to ensure that there is no implementation errors or vulnerabilities.

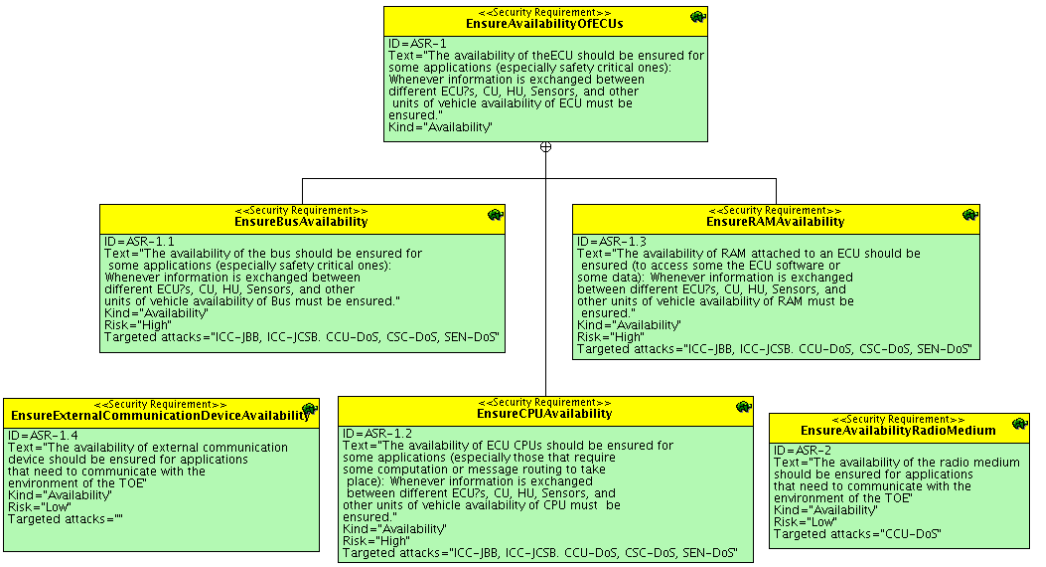

Fig. 5: Security Requirements for Firmware Flashing Process

We examined the system from three points of view: use cases, attack tree, and mapping of functional and architecture design. Merging the result of our analysis from these three directions ensures that security requirements are sufficiently comprehensive to support subsequent design activities. Due to the dependencies highlighted by our approach the elicited security requirements can also be refined at later stages of system development.

5 Conclusion and Future Work

In this paper, we have presented a framework that identifies and models attacks and the most relevant security requirements. We have proposed a model that constructs functional, architectural and mapping views from use case descriptions to clarify a detail set

of attacks and to obtain finer grained system-oriented security requirements. In order to provide OMG conformity throughout system design, we have modeled attacks and security requirements in SysML. At the same time, security requirements are semi-automatically linked with attacks in order to trace the origin of security requirements. Our approach to design follows an iterative process for evaluating, attacks, requirements, at different stages of system development. We have integrated threat modelling, semi-formal security requirements, semi-automated refinement, the Y-chart approach to hardware/software co-design, formal verification, and code generation, all into one environment (TTool).
In addition to identifying, modelling security requirements, and requirement traceability is one area which we are actively investigating and which is missing in text-based approaches. Providing a rationale about the definition of fine-grained requirements helps to understand whether some requirement is still necessary if some assumption about the environment, an attacker, or even the system architecture changes. Another reason with tracing the origin of some requirement stems from the fact that system design is generally architecture and function driven, whereas security is generally non-functional and introduced orthogonally to components.
Assessing the risk associated with attacks constitutes another area for future work. One possible approach is to consider the severity of possible outcome of an attack for the stakeholder and its probability to be successfully mounted. The severity of attacks can be considered in terms of operational, safety, privacy, and financial aspects. In addition to risk analysis and traceability, we are also focusing on security patterns while specifying security requirements.

6 ACKNOWLEDGEMENT

This work has been carried out in the EVITA (E-safety vehicle intrusion protected applications) project, funded by the European Commission within the Seventh Framework Programme (FP7), for research and technological development.

References

[ABD+06] Amer Aijaz, Bernd Bochow, Florian Dötzer, Andreas Festag, Matthias Gerlach, Rainer Kroh, and Tim Leinmüller. Attacks on inter vehicle communication systems - an analysis. *In Proc. WIT*, 189–194, 2006.

[ACLdSS04] L. Apvrille, Jean-Pierre Courtiat, Christophe Lohr, and Pierre de Saqui-Sannes. Turtle: A real-time uml profile supported by a formal validation toolkit. *IEEE transactions on Software Engineering*, pages 473–487, 2004.

[AMAB+06] 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. ICECS 06*, pages 1272–1275, 2006.

[Bal06] Laurent Balmeli. An overview of the systems modeling language for product and systems development - part 1 requirements, use-case, and test-case modeling. Technical report, T.J. Watson Research Center and Tokyo Research Laboratory, IBM, Software Group, 2006.

[CC] *Common Criteria Sponsoring Organizations, Common Criteria for Information Technology Security Evaluation part 1: Introduction and General Model , version 3.1, rev 1, Nat'l Inst of Standards and Technology CCMB-2006-09-001, Sept 2006.*

[EVI] EVITA. E-safety vehicle intrusion protected applications http://www.evita-project.org.

[GMMZ05] Paolo Giorgini, Fabio Massacci, John Mylopoulos, and Nicola Zannone. Sttool: A case tool for security requirements engineering. *13th International Conference on Requirements Engineering, IEEE*, 2005.

[GSWY04] Dan Gordon, Ted Stehney, Neha Wattas, and Eugene Yu. Square methodology: Case study on asset management system phase ii. Technical report, Software Engineering Institute, Carnegie Mellon University, 2004.

[HF04] W. Heaven and A. Finkelstein. A uml profile to support requirements engineering with kaos. *IEE Proceedings, 2004.*, 2004.

[HLMN08] Charles Haley, Robin Laney, Jonathan Moffett, and Bashar Nuseibeh. Security requirements engineering: A framework for representation and analysis. *published in IEEE Trans, Software Eng.(TSE)*, pages 133–153, 2008.

[IEEa] *IEEE Guide to the Software Engineering Body of Knowledge, http://www.swebok.org/.*

[IEEb] *IEEE Standard Glossary of Software Engineering Terminology, Los Alamitos, CA:IEEE Computer Society Press, 1990.*

[ISO] *ISO/IEC 15408, Information technology - security techniques - Evaluation criteria for IT security.*

[Jür02] Jan Jürjens. Umlsec: Extending uml for secure systems development. *5th International Conference on the Unified Modeling Language*, pages 412–425, 2002.

[KFM+09] Enno Kelling, Michael Friedewald, Marc Menzel, Herve Seudie, and Benjaman Weyl. Evita - d: 2.1 specification and evaluation of e-security relevant use cases. Technical report, 2009.

[KMS06] Frank Kargl, Zhendong Ma, and Elmar Schoch. Security engineering for vanets. *4th Workshop on Embedded Security in Cars (escar 2006)*, 2006.

[LBD02] T. Lodderstedt, D.A. Basin, and J. Doser. Secureuml: A uml-based modeling language for model driven security. *5th International Conference on the Unified Modeling Language*, pages 426–441, 2002.

[LvdDV99] Paul Lieverse, Pieter van derWolf, Ed Deprettere1, and Kees Vissers2. A methodology for architecture exploration of heterogeneous signal processing systems. *Workshop on Signal Processing Systems, IEEE*, 1999.

[MGI05] Nancy R. Mead, Dan Gordon, and Theodore R. Stehney II. Security quality requirements engineering (square) methodology, technical report(cmu/sei-2005-tr-009). Technical report, Software Engineering Institute, Carnegie Mellon University,, 2005.

[MI05] Nancy R. Mead and Theodore R. Stehney II. Security quality requirements engineering (square) methodology. *SESS'05*, 2005.

[MIS] *MISRA guidelines for safety analysis of vehicle based programmable systems, ISBN 978 0 9524156 5 7 , MIRA, 2007.*

[PGH06] Panos Papadimitratos, Virgil Gligor, and Jean-Pierre Hubaux. Securing vehicular communications - assumptions, requirements, and principles. *4th Workshop on Embedded Security in Cars (ESCAR 2006), Berlin, Germany*, 2006.

[PS] Cynthia Phillips and Laura Painton Swiler. A graph-based system for network vulnerability analysis. *In Proc. New Security Paradigms Workshop*, 1998:71–79.

[RH07] Maxim Raya and Jean-Pierre Hubaux. Securing vehicular ad hoc networks. *Journal of Computer Security, Special Issue on Security of Ad Hoc and Sensor Networks*, pages 39 – 68, 2007.

[RWW+09] Alastair Ruddle, David Ward, Benjamin Weyl, Yves Roudier, M.s Idrees, Andreas Fuchs, Sigrid Gurgens, Olaf Henniger, Roland Rieke, Michael Friedewald, Timo Leimbach, Ludovic Apvrille, Renaud Pacalet, and Gabriel Pedroza. Evita - d: 2.3 security requirements for automotive on-board networks based on dark-side scenarios. Technical report, 2009.

[Sch99] B. Schneier. Attack trees. *Dr. Dobb's Journal*, 1999.

[SS02] Jan Stefan and Markus Schumacher. Collaborative attack modeling. *In Proc. SAC 2002, ACM*, pages 253–259, 2002.

[Sys] *SysML 1.0 Specification, (ptc/06-05-04), OMG final adopted specification, http://www.omgsysml.org/.*

[TTo] TTool. The turtle toolkit. *http://labsoc.comelec.enst.fr/turtle/ttool.html.*

[UML] *The Unified Modelling Language (UML), http://www.uml.org/.*

[vL04] Axel van Lamsweerde. Elaborating security requirements by construction of intentional anti-models. *ICSE '04: Proceedings of the 26th International Conference on Software Engineering*, pages 148–157, 2004.

[WJY09] Sun Wenhui, Wu Jinzhao, and Xiong YuQing. Methodological support for service-oriented design with rcos. *International Symposium on Information Engineering and Electronic Commerce, IEEE.*, 2009.