SysML-Sec: A Model-Driven Environment for Developing Secure Embedded Systems

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

Abstract: We introduce SysML-Sec, a new SysML environment aimed at making security experts collaborate with system designers at all methodological stages of the design and development of an embedded system. SysML-Sec is also meant to support the assessment of the impact of security over safety. Security and safety concerns are captured in extended SysML diagrams elaborated according to an iterative process centered around the software/hardware partitioning of the architecture. The requirements captured are derived into security and cryptographic mechanisms as well as into security properties that can be formally verified.

Keywords: SysML, security, embedded systems, model driven engineering.

1 Introduction

Most contributions around Model Driven Engineering (MDE) now offer appropriate methodologies and modeling environments for designing safe, complex, distributed, and real-time embedded systems. The analysis of timing constraints, scheduling, resource allocation, and concurrency are commonly handled by these environments. In contrast, security has long been considered only in retrospect, especially after serious flaws are discovered. Security issues have in particular only recently become a major concern in embedded systems. However, the size, heterogeneity, and communication features of modern embedded systems make it compelling to develop a suitable engineering environment to more explicitly define security objectives and threats and to implement countermeasures. The system complexity also makes it worthwhile verifying that requirements are consistent with and satisfied by a candidate design before any commitment to a particular implementation is made.

Contributions in the field of security-aware MDE commonly address one specific methodological stage (e.g., requirements [NNY10], verification [Tou93]), or one specific application domain (e.g., proofs over cryptographic protocols [ABB+05] [ORR00]), or are focused on the immediate modeling of security mechanisms [Jür02] rather than the definition of these mechanisms based on a clearly defined methodology.

This paper introduces SysML-Sec, a new SysML environment with a more holistic approach, which introduces both customized SysML diagrams for security matters and

* Institut Mines-Telecom, Telecom ParisTech, CNRS LTCI, Campus SophiaTech, CS 50193, 06904 Sophia Antipolis cedex France, Tel: 04.93.00.84.06, Fax: 04.93.00.82.00

† EURECOM, Campus SophiaTech, CS 50193, 06904 Sophia Antipolis cedex France, Tel: 04.93.00.81.18, Fax: 04.93.00.82.00

an associated methodology. We intend SysML-Sec to make it possible for security experts to intervene on the design and development of an embedded system together with system designers. The SysML-Sec methodology and diagrams have been developed and experimented in the scope of the FP7 European project EVITA. EVITA defines a secure architecture for automotive embedded systems. The definition, design and validation of this architecture was performed with the methodology which is presented in this paper. Thus, more than 20 use cases (notably an emergency braking case) were taken into account for that purpose, and the diagrams in this paper are directly excerpted from this case study.

2 Hardware / Software partitioning in embedded systems

Software-centric systems are commonly designed with a V-cycle, with building stages (requirements, analysis, design, implementation) followed with verification stages (e.g., tests, formal proofs). For embedded systems, the V-cycle can obviously start only once functions have been partitioned into software and hardware. System partitioning usually relies on the Y-chart approach [BWH+03]. The result of this process is an optimal hardware / software architecture with regards to criteria at stake for that particular system (e.g., cost, performance, etc.) and comprising:

1. *Applications* are first described as abstract communicating tasks: tasks represent functions independently from their implementation form.

2. Hardware *architectures* are described as 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* model defines how tasks and communications between tasks are assigned to computation and communication / storage elements, 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.

We have already defined several MDE-based environment to support the development of embedded systems (the DIPLODOCUS UML profile [AMAB+06], the AVATAR [ADSS11] SysML environment, and the TEPE [KAD11] environment). All those environments are implemented within the free software TTool [Apv13]. TTool automates the formal verification and simulation of models and provides live feedback to UML diagrams. SysML can be used with those environments and tooling to describe the partitioning issues discussed above together with performance and safety requirements.

Security issues are however not addressed by these profiles. We designed the SysML-Sec environment in order to make it possible to describe such issues together with partitioning requirements, as further discussed in [RIA13]. In particular, our extensions bridge the gap between goal-oriented descriptions of security requirements and attacks, and the fine-grained representation of assets based on the software / hardware architecture (and their model-driven analysis).

3 The SysML-Sec approach

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. For example, attacks have been shown to be possible on set-top boxes like Microsoft's XBox [Hua02] or ADSL routers [Ass12], mobile appliances [Ess11], avionics [Tes13], or automotive systems [HKD11] to cite but a few. Many of these security issues reflect either the exploitation of low-level vulnerabilities, which might often be addressed with appropriate programming practices and specific component tests, or design flaws due to an insufficient understanding of the mapping of functional or security logical components to the hardware architecture. We claim that the SysML-Sec Model-Driven Engineering approach makes it possible to perform an appropriate system analysis in both directions, and to describe both security threats and security objectives by:

- Guiding and increasing the collaboration between system engineers and security experts throughout the entire embedded system lifecycle. This has been the reason for our adoption of the OMG standards, and more specifically SysML, which are quite widespread in the embedded system world today.

- Providing detailed representations of the security threats and security requirements compatible with the MDE methodology used and making it possible to adopt a stepwise refinement approach to the definition of both the functional and the security architecture. This refinement should also make it possible to bridge the gap between initial high-level requirements and the definition of precise and detailed security mechanisms.

- Combining software/hardware codesign together with the handling of security concerns. We contend that this particular design objective is a key in the embedded system domain.

The SysML-Sec methodology adopts a three-phase approach that first deals with the system analysis, then with software design, and finally with system validation, as described in the following sections.

4 System requirement engineering and analysis

The security requirement and threat analysis is mostly regarded 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 means additional costs. In the case of embedded systems, we contend that the security analysis also has a strong impact on the system architecture and its realtime performance: the security requirements and threat analysis should thus be performed along the partitioning iterative process. We also claim that the security analysis should 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 and to the risk model.

4.1 Iterative security/system codesign process

System partitioning, security requirements, and threats are progressively refined based on one or several typical use cases. The following phases, which thus start with an initial architecture, are iterated in order to reach a satisfactory level of refinement:

Initial architecture mapping. The functionalities of the system highlighted in these use cases are first modeled as tasks. Exchanges between functions 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 the first draft of the architecture.

Architecture analysis *System assets* are identified among architectural elements (processors, pieces of software, sensors, hardware accelerators, communication channels) 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. The hardware/software partitioning and the function mapping adopted play a key role here in defining the type of asset at hand (and later on its vulnerabilities).

Security concern identification. *Threats and security vulnerabilities of the selected assets* should as much as possible describe the capabilities that an attacker should meet or exceed and the origin of attacks (local, remote, through a specific interface). The SysML-Sec environment supports the assessment of risks following the approach described in more detail in the EVITA case study [Rea09, HAF+]. We also implemented automated checks of the threat coverage by security objectives. Based on the risk analysis, one should also identify and prioritize security objectives that are mapped to a threat. *Security objectives* might originate (1) from security standards or properties expected from the system, or (2) from unaddressed threats or attacks on assets, or (3) from the refinement of another security objective when the process is iterated and the level of detail of the architecture has changed. In further iterations, one may need to update security objectives deprecated by changes in the architecture.

Architecture refinement. The architecture refinement originates from a more detailed description of the architecture components as the system and its usage become more precisely known (e.g., new communication channels, refinement of an execution environment into OS/middleware/application layers, etc.). It may also result from transitively mapping requirements to system information flows, which are often distributed among multiple hardware elements. 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 stage of refinement.

4.2 Diagrams

In our proposed framework, the partitioning is given using the allocations of tasks over hardware nodes. Tasks and hardware nodes are modeled using SysML blocks. Allocations are modeled with the SysML "allocate" relationship.

Security requirements are modeled in SysML Requirement Diagrams (RD). The main operators of RDs are *Requirement Containment* and *Derive Dependency* formalisms used

to define relationships between requirements. The *containment* relationship depicts 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 in the requirement hierarchy. A *Security Requirement* stereotype is introduced to make a clear distinction between functional requirements and security requirements of the system, yet modeling both functional and non-functional requirements in a single environment. Furthermore, a *Kind* parameter is defined to specify the category of the security requirement (*confidentiality*, *access control*, *integrity*, *freshness*, etc.).

Attack trees can be modeled with slightly customized SysML Parametric Diagrams. Attacks are modeled as values embedded into blocks representing the target of the attack. Attacks can be linked together with logical operators like OR, AND, as well as temporal causality operators like $AFTER$, the latter which we consider as especially helpful to describe attacks in embedded systems. Their instances in different parametric diagrams can be linked together in order to assess the impact of a specific vulnerability and the need to address it at the risk assessment phase. An attack can also be tagged as a *root* attack, meaning that this attack is at the top of the tree. Last but not least, attacks can be linked to requirements, thus allowing an automated check of the coverage of attacks.

An example of function mapping is given in Figure 1, made in the scope of the emergency braking (or Local Danger Warning) use case. Two Electronic Control Unit 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.

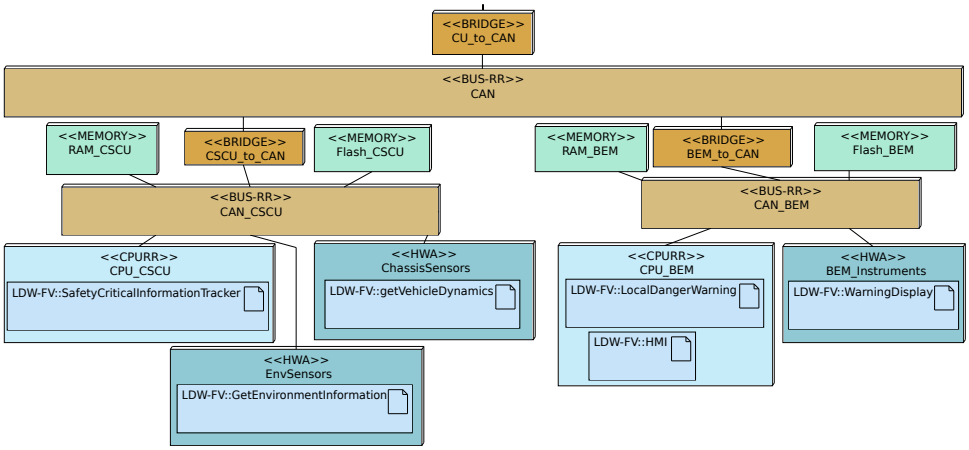

Fig. 1: Mapping of Local Danger Warning use case (SysML-Sec)

5 Software design

5.1 Methodological aspects

Software design defines the architecture and behaviour of all functions mapped over processor nodes at the partitioning stage. From a security point of view, the design intends to precise how security requirements can be fulfilled with security-oriented software mechanisms executed on top of the hardware architecture defined in the partitioning stage, and verify that requirements identified during the partitioning phase are really satisfied by this design. Requirements expressed at partitioning are informal and refer to assets: they therefore need to be refined until their expression directly relates to design elements (e.g., attributes, methods, exchanged messages, states, etc.). Once refined, they constitute the security properties that are to be verified by the design. SysML-Sec extends SysML with ways to explicitly model security mechanisms and properties.

5.2 Security design extensions

A SysML-Sec design is made upon SysML block and state machine diagrams, extended with several features, and formally defined in pi-calculus (a process algebra).
In particular, we assume a Dolev-Yao attacker model, that is, only messages exchanged between blocks can be eavesdropped, contrary to attributes of blocks. That attacker model is enough to describe attacks on the protocols deployed between the components of the embedded system, from outside or within the system. It however does not aim at capturing physical attacks on the hardware, nor a sequence of exploitation of vulnerabilities of several components. Since communication channels may have been mapped over secure or non secure buses at partitioning stage, we give the possibility to tag links between blocks with a *public* label if an attacker can eavesdrop, or with a *private* label otherwise.
Also, SysML-Sec blocks can define a set of methods corresponding to cryptographic algorithms (e.g., *encrypt*), so as to be able to describe security mechanisms built upon these algorithms, e.g., cryptographic protocols. Blocks can also pre-share values, a feature commonly needed to setup cryptographic protocols. SysML-Sec introduces specific *pragmas* for that purpose (*InitialSystemKnowledge* and *InitialSessionKnowledge*).

5.3 Security properties

A dedicated language has been defined for describing the commonly complex safety properties, which is based on SysML Parametric diagrams [KAD11]. On the contrary, security properties can usually be defined with a type (e.g., *confidentiality*), and with design elements related to that kind (e.g., the confidentiality of the attribute of a block). This simplicity pleads for a basic modeling solution, that is not based on complex diagrams or operators. Our solution relies on *pragmas* provided in notes of Block Diagrams: *confidentiality* and *authenticity* can be directly expressed at this level.

For example, an authenticity pragma states that a message $m2$ received by a block *block*2 was necessarily sent before in a message $m1$ by a block *block*1. The following examples describes such a situation: `♯ Authenticity block1.s1.m1 block2.s2.m2`
This authenticity pragma specifies two states: one of the sender block, i.e. one state $s1$ of *block*1, and one state $s2$ of *block*2. Also, in the state machine diagram of *block*1, $s1$ corresponds to the state right before the sending of $m1$. Analogously, $s2$ corresponds to the state right after message $m2$ has been received and accepted as authentic.

Figure 2 illustrates the SysML-Sec software design of the Key Distribution protocol defined in the EVITA architecture. This protocol distributes session keys in a group of Electronic Control Units (ECUs) spread in the vehicle. Pragmas are given in the top left part of the diagram: the knowledge which is pre-shared between ECUs and the Key Master (KM), and two properties: the confidentiality of the key pre-shared with ECU1, and the authenticity of a message sent from ECU1 to KM. The blocks underneath model the ECU initiating the key distribution (ECU1), the Key Master, and another ECU (ECUN).

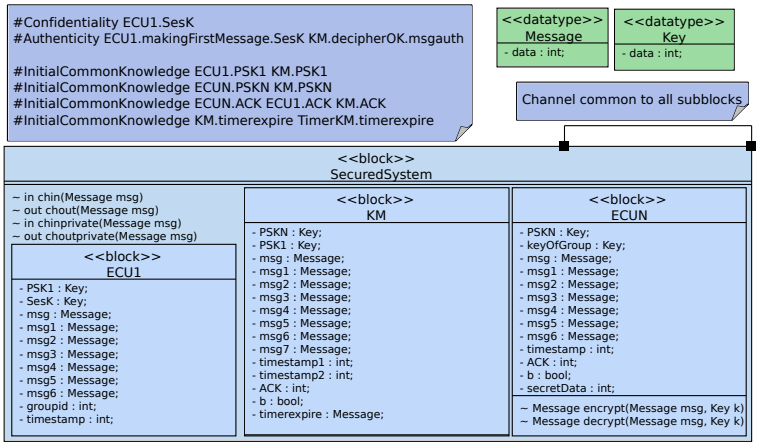

Fig. 2: SysML-Sec block diagram of Key Distribution Protocol

6 System validation

Validation can be performed from mapping models (e.g., performance evaluation of the selected hardware architecture: load of CPUs and buses), from design models (proof of safety and security properties), or from executable code automatically generated from design models (safety and security tests). Model transformations have been defined to transform SysML-Sec models into formal specifications. The whole process is seamlessly implemented in TTool, i.e., a user of TTool does not need to know about underlying formal techniques since model transformations and backtracing to models is totally automated. Proofs can be performed from partitioning or software design models. From partitioning models, it is possible to evaluate the impact of security mechanisms onto real-time constraints (e.g., latencies). From SysML-Sec designs, the formal proof relies on *ProVerif* for security properties. *ProVerif* [Bla09] is a toolkit that relies on Horn clauses resolution for the automated analysis of security properties over cryptographic protocols, under the Dolev-Yao model. ProVerif takes in input a set of Horn Clauses, or a specification in pi-calculus and a set of queries. ProVerif outputs whether each query is satisfied or not. In the latter case, ProVerif tries to identify a trace explaining how it came to the conclusion that a query is not satisfied. Figure 3 depicts the successful verification in TTool of the confidentiality and authenticity properties modeled in Figure 2. While we can specify any security requirement, we currently only support the formal validation of those that can be expressed with these two security properties.

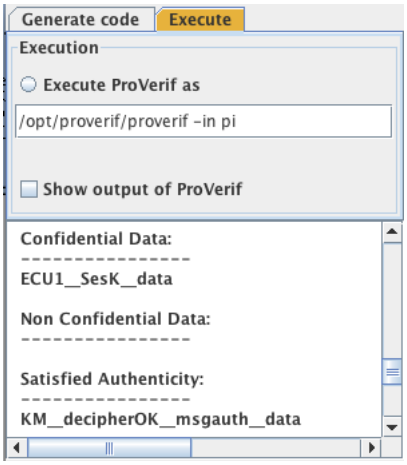

Fig. 3: TTool assistant for the formal verification of confidentiality and authenticity properties defined at Figure 2

7 Related work

In [NNY10], Nhlabatsi et al. classify security requirements engineering work in software systems according to four dimensions, namely: (1) *goal-based approaches* was the first such approach to model, specify, and analyze security requirements as the organized refinement of goals into a set of sub-goals using generic patterns. (2) *Model-based approaches*: in contrast, those methodologies focus on the mapping of security to the software architecture. (3) *problem-oriented approaches* focus on the expression of attacks and on the threat-based elicitation of security requirements (abuse frames, misuse cases). (4) *Process-oriented approaches* finally aim mainly at the risk analysis of an existing design and follow a rather rigid waterfall approach to engineering, yet do not address well design exploration and refinement. Our approach combines a goal-based description of security requirements with a model-driven engineering of the system architecture and validation of the soundness of the security properties or of their innocuity with respect to safety. In that respect, it shares some similarities with the TwinPeaks approach advocated by Nuseibeh [Nus01], although the latter doe not address hardware 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.

UMLsec [Jür02], which features a model-based approach as described above, defines how to integrate security protocol descriptions and security properties to a UML framework. However, design elements and security properties are mixed on the same diagrams, as well as functional and non functional requirements.

Assessing security in embedded systems mostly relies on formal approaches. For example, [Tou93] proposes to verify cryptographic protocols with a probabilistic analysis approach. In more recent efforts, [DBTS] embeds a first order Linear Temporal Logic (LTL) in the Isabelle/HOL theorem prover, thus making it possible to model both a system and its security properties, but unfortunately leading to non-easily reusable specific

models. [MP08] mixes formal and informal security properties, but the overall verification process is not completely automated, again requiring specific skills.

8 Conclusion and future work

The complexity of embedded systems and time-to-market and software-engineering constraints plead for engineering security requirements with user-oriented tools featuring automated and simplified verification. Our proposal, SysML-Sec, specifically addresses that need at the diverse phases of system design and development. It is based on a popular and friendly language (OMG's SysML) and supported by an open-source toolkit (TTool) that relies on a recognized security verification toolkit (ProVerif). We have also developed a methodology for the use of SysML-Sec diagrams which has been experimented to define a secure automotive embedded system.

References

[ABB+05] Alessandro Armando, David A. Basin, Yohan Boichut, Yannick Chevalier, Luca Compagna, Jorge Cuéllar, Paul Hankes Drielsma, Pierre-Cyrille Héam, Olga Kouchnarenko, Jacopo Mantovani, Sebastian Mödersheim, David von Oheimb, Michaël Rusinowitch, Judson Santiago, Mathieu Turuani, Luca Viganò, and Laurent Vigneron. The AVISPA tool for the automated validation of internet security protocols and applications. In *CAV*, pages 281–285, 2005.

[ADSS11] L. Apvrille and P. De Saqui Sannes. AVATAR/TTool : un environnement en mode libre pour SysML temps réel. *Génie Logiciel*, (98):22–26, September 2011.

[AMAB+06] L. Apvrille, W. Muhammad, R. Ameur-Boulifa, S. Coudert, and R. Pacalet. A UML-Based Environment for System Design Space Exploration. In *Electronics, Circuits and Systems, 2006. ICECS '06. 13th IEEE International Conference on*, pages 1272 –1275, Dec. 2006.

[Apv13] Ludovic Apvrille. TTool website. In *http://ttool.telecom-paristech.fr/*, 2013.

[Ass12] Fabio Assolini. The Tale of One Thousand and One DSL Modems, kaspersky lab, October 2012.

[Bla09] B. Blanchet. Automatic Verification of Correspondences for Security Protocols. *Journal of Computer Security*, 17(4):363–434, July 2009.

[BWH+03] F. Balarin, Y. Watanabe, H. Hsieh, L. Lavagno, C. Passerone, and A. Sangiovanni-Vincentelli. Metropolis: An Integrated Electronic System Design Environment. *Computer*, 36(4):45–52, April 2003.

[DBTS] Michael Drouineaud, Maksym Bortin, Paolo Torrini, and Karsten Sohr. A first step towards formal verification of security policy properties for rbac. In *Proceedings of the Quality Software, Fourth International Conference*, QSIC'04, pages 60–67, Washington, DC, USA. IEEE Computer Society.

[Ess11] Stefan Esser. iOS Kernel Exploitation. In *BlackHat 2011*, 2011.

[HAF+] Olaf Henniger, Ludovic Apvrille, Andreas Fuchs, Yves Roudier, Alastair Ruddle, and Benjamin Weyl. Security Requirements for Automotive On-Board Networks. In *Proceedings of the 9th International Conference on Intelligent Transport System Telecommunications (ITST 2009), Lille, France.*

[HKD11] Tobias Hoppe, Stefan Kiltz, and Jana Dittmann. Security Threats to Automotive CAN Networks - Practical Examples and Selected Short-Term Countermeasures. *Rel. Eng. & Sys. Safety*, 96(1):11–25, 2011.

[Hua02] Andrew Huang. Keeping Secrets in Hardware: the Microsoft XBox Case Study, AI Memo 2002-008, Massachusetts Institute of Technology, Artificial Intelligence Laboratory. Technical report, 2002.

[Jür02] Jan Jürjens. UMLsec: Extending UML for Secure Systems Development. *5th International Conference on the Unified Modeling Language*, pages 412–425, 2002.

[KAD11] D. Knorreck, L. Apvrille, and P. De Saqui-Sannes. TEPE: A SysML Language for Time-Constrained Property Modeling and Formal Verification. *ACM SIGSOFT Software Engineering Notes*, 36(1):1–8, January 2011.

[MP08] Antonio Maña and Gimena Pujol. Towards formal specification of abstract security properties. In *The Third International Conference on Availability, Reliability and Security*, volume 0-7695-3102-4/08. IEEE, 2008.

[NNY10] Armstrong Nhlabatsi, Bashar Nuseibeh, and Yijun Yu. Security Requirements Engineering for Evolving Software Systems: a survey. Technical Report 1, The Open University, 2010.

[Nus01] Bashar Nuseibeh. Weaving Together Requirements and Architectures. *IEEE Computer*, 34(3):115–117, 2001.

[ORR00] Peter Ochsenschläger, Jürgen Repp, and Roland Rieke. The SH-Verification Tool. In *Proceedings of the Thirteenth International Florida Artificial Intelligence Research Society Conference*, pages 18–22. AAAI Press, 2000.

[Rea09] A. Ruddle and et al. Security Requirements for Automotive On-board Networks Based on Dark-side Scenarios. Technical Report Deliverable D2.3, EVITA Project, 2009.

[RIA13] Yves Roudier, Muhammad Sabir Idrees, and Ludovic Apvrille. Towards the model-driven engineering of security requirements for embedded systems. In *proceedings of the 3rd International Model-Driven Requirements Engineering (MoDRE) workshop*, July 2013.

[Tes13] Hugo Teso. Aircraft Hacking. In *HITB Security Conference*, Amsterdam, The Netherlands, 2013.

[Tou93] M. J. Toussaint. A New Method for Analyzing the Security of Cryptographic Protocols. In *Journal on Selected Areas in Communications*, volume 11, No. 5. IEEE, June 1993.