SysML-Sec
A model Driven Approach for Designing Safe and Secure Systems

Yves Roudier[1] and Ludovic Apvrille[2]
[1]*EURECOM, 450 Routes des Chappes, 06410 Biot Sophia-Antipolis, France*
[2]*Institut Mines-Telecom, Telecom ParisTech, CNRS LTCI*
450 Routes des Chappes, 06410 Biot Sophia-Antipolis, France
yves.roudier@eurecom.fr, ludovic.apvrille@telecom-paristech.fr

Keywords: Model-driven engineering, safety, security, SysML, embedded systems.

Abstract: Security flaws are open doors to attack embedded systems and must be carefully assessed in order to determine threats to safety and security. Subsequently securing a system, that is, integrating security mechanisms into the system's architecture can itself impact the system's safety, for instance deadlines could be missed due to an increase in computations and communications latencies. SysML-Sec addresses these issues with a model-driven approach that promotes the collaboration between system designers and security experts at all design and development stages, e.g., requirements, attacks, partitioning, design, and validation. A central point of SysML-Sec is its partitioning stage during which safety-related and security-related functions are explored jointly and iteratively with regards to requirements and attacks. Once partitioned, the system is designed in terms of system's functions and security mechanisms, and formally verified from both the safety and the security perspectives. Our paper illustrates the whole methodology with the evaluation of a security mechanism added to an existing automotive system.

1 INTRODUCTION

Embedded systems and Cyber-Physical Systems progressively impact our daily lives, information systems, and industrial systems. Because these systems are by essence connected, they are strongly exposed to attacks. To cite only a few examples of past attacks on similar connected systems, we can mention the Microsoft XBox (Huang, 2002), ADSL routers (Assolini, 2012), mobile&smart phones (Maslennikov, 2010) (Esser, 2011) (Apvrille and Strazzere, 2012), avionic systems (Teso, 2013) or automotive systems (Teso, 2013), and even home appliances such as refrigerators as demonstrated recently by a botnet infecting LG fridges (Proofpoint, 2014). Such attacks also target industrial systems whose sensors are more and more commonly connected with vulnerable information systems, as demonstrated by the Stuxnet, Flame, or Duqu (Maynor, 2006) attacks. The dependability of such systems can of course be targeted by such attacks with very various objectives, e.g., terrorist acts and ransomware.

The complexity of such systems in terms of code size, distribution, and heterogeneity among others is a major factor for the risks they face. We contend that taking into account security from the very first development phases, and including both software and hardware components, could lower the risk those systems face, both in terms of security and safety. We also think that safety and security should be designed and validated all together, that is, from the same models, and that the impact of the security mechanisms execution over the safety-related functions should be clearly established. Our solution relies on a model-driven environment named SysML-Sec. SysML-Sec covers all design and development phases, including requirement capture, attack equations, functional model, hardware/software partitioning, designs of software components, and validation of both safety and security properties. The validation itself shall be performed at partitioning and design stages. In the first case, the objective is to assess the effectiveness of the selected hardware/software architecture to support both safety and security requirements, and resist to identified attacks. In the second case, the validation should closely investigate whether the design choices respect the safety and security requirements.

SysML-Sec has already been presented in (Apvrille and Roudier, 2013). This paper focuses on a new aspect, that is, how SysML-Sec can also be used to evaluate the impact of security-oriented design onto systems with strong critical aspects, e.g., critical embedded systems, or systems whose attack may lead to strong economical loss, e.g., new interconnected information systems such as cloud services.

The paper is organized as follows. Section 2 focuses on the impact of attacks on safety-critical systems. Section 3 discusses different modeling approaches to efficiently take into account both the system architecture and requirements in the development process of a safe ad secure system. Then, we present the methodology and the validation process of the SysML-Sec framework in Sections 4 and 5, respectively, with a strong emphasis on the relation between safety and security properties and features. An automative-based case study is given in section 6. Section 7 concludes the paper.

2 IMPACT OF ATTACKS

Performing attacks relies on the exploitation of either low-level vulnerabilities (e.g., buffer overflows) or design weaknesses. Low-level vulnerabilities can often be handled with good programming practices, or by using efficient security tests, e.g., different kinds of fuzzing. On the contrary, design errors generally come from a bad requirement and attack capture, or from a bad partitioning of hardware / software components, including security mechanisms that those components implement. Unfortunately, those errors are more difficult to correct once the system has been released. Moreover, security tools commonly used by security experts, e.g. ProVerif (Blanchet, 2009) and AVISPA (Armando et al., 2005), not take into account the hardware/software partitioning.

Classical information systems are still widely targeted by attackers. Nonetheless, the particularities of information systems comprising communicating embedded equipments make them even more "interesting" targets for attackers. We can mention three dangers related to those attacks:

- If an attack relies on a hardware vulnerability, it can be difficult, or even impossible, to correct (i.e., patch) the system via a software update, in particular if functions have been implemented mostly with hardware components, e.g., with hardware accelerators or network filtering techniques. In the case of usual information systems, a "simple" software update and a system reboot can generally solve the problem.
- Updating the software of embedded system is much more complex than updating the software of, e.g., PCs. For instance, to patch the software of a vehicle, it is necessary to send a letter to the owner, that will later bring the car to a maintenance garage. Users of embedded systems are also less used to update their systems, as recently demonstrated with IP cameras. On the contrary, updating a PC or a server has become a common practice (the famous "Tuesday patch"), and can be handled remotely in an automated way.
- The impact of attacks on safety critical systems can be much higher than on other kinds of systems. Moreover, the growing interconnection between information systems and smart objects gives more chances to attackers to enter illegally those systems, once he/she has managed to compromise one of the interconnected element. This is the case, e.g., for botnets (DoS attacks, spams), as demonstrated recently by the LG fridge attack (Proofpoint, 2014), or for performing cryptographic attacks.

3 THE FUNDAMENTAL ROLE OF SYSTEM ARCHITECTURE TO DEVELOP A SECURE SYSTEM

From our own experience, we think that main system architectural elements must be first modeled in order to better capture and analyze potential attacks on the system.

3.1 Software/Hardware partitioning

Software-centric systems are commonly designed with a V-cycle comprising the following stages: requirements elicitation, software analysis, software design, implementation. Each of these stages are then followed with a corresponding verification state, that commonly relies on testing, simulation and formal verification techniques. In the case of embedded systems, that approach is obviously applicable only once functions to be software implemented have been specified. In other words, the V-cycle can start only once the software and hardware partitioning has been performed.

System partitioning (a.k.a. Design Space Exploration) is a process to analyze various functionally equivalent implementation of systems specification. It usually relies on the Y-chart approach (Balarin et al., 2003) depicted in Figure 1:

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

2. Targeted *architectures* are independently described from tasks. They are usually described with a set of execution nodes (e.g., CPU), communication nodes (e.g., buses), and storage nodes (e.g., memories).

3. A *mapping* model defines how application tasks and abstract communications 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.

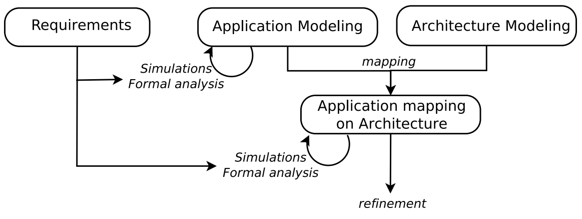


Figure 1: The Y-chart methodology

Ideally, the result of the Y-chart approach shall be an optimal hardware / software architecture with regards to criteria at stake for that particular system (e.g., cost, area, power, performance, flexibility, reliability, etc.). This process can be manually performed by experiemnted engineers, or can be automated by specific toolkits, e.g. (Balarin et al., 2003). This partitioning step 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.

3.2 Dimensioning

In some highly critical systems, models for design space exploration are not intended for computing and analyzing *Worst Case Execution Time*. On the contrary, dimensioning-based techniques based on network calculus (Leboudec and Thiran, 2001) are adapted to compute worst case execution scenarios. These techniques are typically used for defining the architecture of aeronautics systems. Dimensioning models are part of our overall toolkit (Apvrille et al., 2010), even if they are not (yet) used in SysML-Sec.

3.3 Model-driven engineering

Model-Driven Engineering is probably the main contribution of the last decade in modeling approaches. MDE targets system analysis, design, simulation, code generation, and documentation. MDE generally relies on the UML language, and on meta-modeling techniques in order to define Domain-Specific Languages. OMG's Model Driven Architecture for instance specifically targets two abstraction levels, namely the Platform-Independent Model (PIM) and Platform Specific Model (PSM): embedded systems are thus clearly targeted by MDE. Profiles have also been defined by the OMG to more specifically address embedded systems: SPT (OMG, 2005) and MARTE (Vidal et al., 2009), but none of them addresses requirements modeling. Conversely, the SysML OMG profile (OMG, 2012) clearly takes into account requirements with explicit modeling capabilities and diagrams, but ignores some problematics inherent to embedded systems, e.g., the partitioning issue.

Other methodologies, like for example *Extreme programming* (XP) (Beck and Andres, 2004) or *Agile Software Development* (Waters, 2012) have also been proposed to develop software-oriented systems. However, their software focus means that they totally ignore the partitioning issue. They also make traceability and refinement extremely hard to achieve, also because requirements are mostly separate from the design process.

4 SPECIFYING THE SECURITY NEEDS OF A SYSTEM

Security and privacy are commonly after-thought in new connected and distributed information systems (e.g., cloud services) and embedded system, in contrary to safety. Thus, security needs appear after the system has already been released, generally when security vulnerabilities are discovered. Nonetheless, those vulnerabilities have a critical aspect whenever the can be exploited to impact economical or safety-related components.

4.1 Security goals and threats

Many research works have already addressed the modeling and analysis of security requirements and threats, mostly in the scope of information systems. Nhlabatsi et al. (Nhlabatsi et al., 2010) classify them in four categories: goal-oriented approaches, model-driven approaches, process-oriented approaches and last problem-oriented approaches. The two first are the two closest to SysML-Sec. KAOS is a well-known goal-oriented approach (Van Lamsweerde, 2007) that relies on the explicit model of security goals and anti-goals. TwinPeaks (Nuseibeh, 2001) also follows a goal-oriented approach with an agile iterative process between goals and system architecture. UMLSec is a model-oriented approach. It includes tools for the specification and verification of distributed systems with security mechanisms, including cryptographic protocols. Model-oriented approaches are considered as more adapted to the design of security mechanisms, but goal-oriented approaches offers a better way to analyze security requirements during the first design iterations.

4.2 SysML-Sec: A SysML-based model-oriented approach

The first objective of SysML-Sec (Apvrille and Roudier, 2013) is to facilitate the collaboration and communication between experts in system design and experts in system security and privacy. Moreover, SysML-Sec intends to cover all development methodological stages of these system. SysML-Sec combines a goal-oriented approach for capturing requirements, and a model-oriented approach for system architecture and threats. In contrary to TwinPeaks (Nuseibeh, 2001), SysML-Sec follows the Y-chart scheme (Balarin et al., 2003) and its underlying allocation techniques. The latters facilitates the identification of resources to be protected and the link between resources, safety requirements, and security requirements.

4.3 Methodology

The SysML-Sec methodoogy is first based on an Y-Chart-based system analysis, and then on a software design phase following the well-known V-cycle. This methodology is given at Figure 2.

The analysis stage intends to identify and analyze both requirements and attacks altogether with the identification of main functions ("application"), candidate hardware architectures, and the mapping of main functions over execution nodes (mapping). During the functional stage, simulations and (formal) verifications are used in order to identify safety-related issues, e.g., deadlock situations, non -reachability of error states, etc. Functional models are untimed, which means that no performance study can be lead. Yet, the mapping of functions over execution nodes gives to the former a logical and physical execution time. Thus, post-mapping simulations and formal verifications are intended to demonstrate the system performance on the selected hardware architecture, including the study of latencies, the load of processor and buses, and communication time. Obviously, the results are due to the logical parts of the application, to the way the underlying hardware behaves, but also to the security mechanisms. For example, a given security protocol may impact a bus load, a cryptographic function may impact a processor load: Both can consequently increase the overall system latency. The mapping scheme, for example, mapping a cryptographic function over a hardware accelerator, or on a general-purpose processor, also impacts system latencies. The performance study thus really intends to study altogether both safety and security functions mapped in different conditions. The result of this study is a hardware/software architecture that complies with both safety and security requirements, and that can resist to attacks, and according to a given risk level.

The goal of the design stage is to design the software components, that is, functions mapped onto processors at previous stage. During the design, software components are progressively refined until the point where executable code generation is feasible. This refinement also includes security-related functions, e.g., security protocols. During the first refinements, simulation can be used to debug the models. When the model is of reasonable size, formal verification can also be used to assess safety properties (e.g., the reachability of a given state, or its liveness), and security properties (e.g., the confidentiality of a given block attribute, the authenticity of a message). When the model is too large to be verified, model-to-code transformations are used to perform security and safety tests. This paper do not address the design stage, but it is described in (Apvrille and Roudier, 2013).

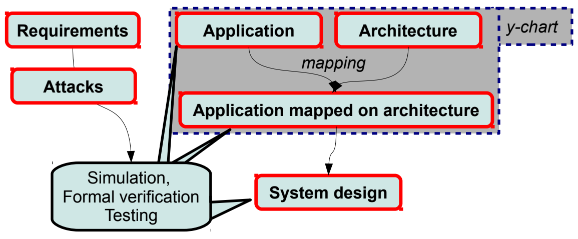


Figure 2: The SysML-Sec methodology

4.4 Toolkit

The free and open-source TTool software supports all SysML-Sec methodological stages, including the models capture and their simulations and verifications. In fact, TTool is a multi-profile toolkit which main strength is to offer a press-button approach for performing simulation and formal proofs from models. Proofs can be performed automatically for both safety and security properties. For the partitioning stage, TTool relies on the DIPLODOCUS UML profile (Apvrille et al., 2006), even if security aspects presented in this paper are specific to SysML-Sec. Requirements, attacks, software design and all captured and analyzed with AVATAR (Apvrille and De Saqui-Sannes, 2013) which covers the V development cycle. Formal verifications and simulations can be performed either with TTool's integrated model checkers and simulators, or with external formal verification toolkits, e.g. UPPAAL (Bengtsson and Yi., 2004), CADP (Garavel et al., 2007) and ProVerif (Blanchet, 2009).

5 SAFETY AND SECURITY ORIENTED VALIDATIONS

Model-oriented approaches do favor the early validation of models. SysML-Sec supports such validation, either from partitioning models, or from design models. Validations are supported by TTool with a press-button approach: models can be transformed by TTool into a formal specifications. TTool has its own model-checker, but it can also relies on external provers. Once the validation results have been obtained (property is satisfied, property is not satisfied, property cannot be proved), the model is back-annotated with the results. Formal validation is more likely to be used to prove specific and precise safety and security property, e.g., that a given data is confidential. Simulation is more likely to be used whenever the impact of a security mechanism over safety properties (e.g., latency-related properties) must be studied. This impact is mostly studied at partitioning stage.

5.1 Safety properties

Such validations concern the following models: functional models, partitioning models, and design models. Properties can be proved with the use of simulation techniques (Apvrille and De Saqui Sannes, 2011), of formal verification techniques (Apvrille and De Saqui-Sannes, 2013) (Bengtsson and Yi., 2004), or by generating an executable code (only from refined design models) and performing tests on that code (Apvrille and Becoulet, 2012). From design diagrams, safety proofs take into account both safety and security mechanisms, because the latter impact the behaviour of software components, e.g., taking a given execution path.

The system architecture and behaviour are commonly modeled with graphical languages. On the contrary, languages used for expressing properties still mostly rely on textual languages, e.g. LTL/CTL, or languages derived from those ones. Yet, it is possible in SysML-Sec to model all logical and timing properties with SysML parametric diagrams (Knorreck et al., 2011).

5.2 Security properties

In the design stage, formal proofs of confidentiality and authenticity properties can be performed. If possible, security requirements must first be refined into confidentiality and authenticity properties modeled as pragmas in SysML-Sec block diagrams. TTool can then transformed the design diagrams into a security-oriented formal specification in pi-calculus (Pedroza, 2013). This specification can be given as input to ProVerif, along with the confidentiality and authenticity properties to be proved (Blanchet, 2009).

5.3 Safety and security, safety vs. security

It is not a common practice to assess the relationship between safety and security at partitioning stage. Yet, we believe that this should be addressed as soon as possible in the development cycle, as suggested by the SysML-Sec methodology. Eames and Moffet (D. P. Eames and Moffett, 1999), and more recently Piètre-Cambacédès (Pietre-Cambacedes and Bouissou, 2013) and Raspotnig (Raspotnig and Opdahl, 2013) have proposed ways to handle the links between safety and security properties. More precisely, (Pietre-Cambacedes and Bouissou, 2013) and (Raspotnig and Opdahl, 2013) aim at explicitly describing conflicts between such requirements, and also to provide "reinforcements", that is, requirements with different scope that drives the architecture models in the same direction.

As we show in the next section, SysML-Sec makes it possible to assess the compatibility of security mechanisms with regards to safety properties in partitioning stage, relying on validation techniques. Similar studies can be lead during the design stage.

During the partitioning stage, mechanisms are evaluated according to the system latency and the usage of the platform, e.g., the necessary computation power, the load of buses, and the respect of real time deadlines. Partitioning models are typically validated by generating a known average traffic in the system, and then introducing security mechanisms so as to evaluate the impact of the latter on the former. The refinement of SysML-Sec models leads to introduce fine-grained cryptographic functions where properties to be proved are more likely to be the reachability of the given state of a security protocol. Thus, at design stage, we also rely on provers so as to assess the consistency of requirements and the coverage of attacks.

6 EXAMPLE: SAFETY AND SECURITY ANALYSIS OF A COMMUNICATING VEHICLE

This section illustrates the use of SysML-Sec for the identification of requirements and attacks and for the definition of safe and secure system architecture. The automotive system that serves as case study is taken from the European FP7 EVITA project (Kelling et al., 2009). EVITA has defined the first generic security architecture for automotive communicating systems. This architecture contains safety critical ECUs (Electronic Control Units) interconnected with CAN or Flexray buses. Automotive systems are likely to be attacked either for economical reasons (activating optional features for free, stealing a car), either for criminal purpose. The interconnection of automotive systems to information systems (roads signs, tolls, etc.) and Internet will offer new ways to conduct attacks onto those systems.

A reference automotive architecture is given in Figure 3. It is built upon several domains (Powertrain, chassis & Safety, Communication Unit, Head Unit, etc.). Domains may contain several sub domains, each of them containing several buses and processors. A main CAN/FlexRay bus interconnects all ECUs bridges together. At last, the communication and head units have external interfaces (Bluetooth, LTE, etc.).

We now apply the SysML-Sec methodology with the addition of a security mechanism to the reference architecture. This mechanism is the distribution of cryptographic session keys between domains and subdomains. An ECU1 asks the “key master” to generate a session key so as to communicate in a secure way with other domains ECUN of this session.

6.1 Security requirements

Security requirements are captured within SysML requirements diagrams. Security requirements are stereotyped $<<$ *SecurityRequirement* $>>$ and contains an extra *kind* field (*confidentiality*, *access control*, *integrity*, *freshness*, etc.). Usual SysML relations between requirements (*containment*, *derive*) can be used, in particular to trace requirements throughout the different abstraction levels. Figure 4 presents an excerpt of security requirements dealing with the prevention of the injection of wrong commands into the system. The main requirement is refined into authenticity, integrity and freshness requirements for all communication between ECUs involved in system functions.

6.2 Threats and attacks

Attacks are commonly captured with attack trees (Schneier, 1999). Such trees allow an efficient decomposition of attacks into sub-attacks. Yet, they are not adapted to the description of complex scenarios. Yet, attacks currently lead on embedded systems are generally composed of complex scenarios, sometimes involving complex logical and timing relations, e.g., the Zeus/Zitmo attack exploits a time-limited authentication token.

We thus decided to introduce a richer attack model based on SysML parametric diagram: *attack equations*. These equations are meant to offer a set of constraints linking together sub-attacks, in order to build more complex attacks.

Attacks are captured with “properties” of SysML blocks. Since Blocks represent resources of the system, a visual emphasis on architectural elements to protect is obtained. Constraints are used to relate attacks together: $<< OR >>$, $<< AND >>$, $<< SEQUENCE >>$, $<< BEFORE >>$, $<< AFTER >>$. Each constraint has a denotational correspondence, e.g. the equation of a $<< OR >>$ between two attacks $a1$ and 2 is $a = a_1 \vee a_2$, with a being the resulting attack. Attacks can also be linked to requirements, and vice-versa.

Figure 5 depicts a simplified attack of our case study. A main resource *AutomotiveECUsandBuses* is the target of an attacker wishing to steal the car (*Steal_car* root attack). To successfully perform that attack, the attacker first need to connect on the system bus so as to be able to open the doors and then start the engine. Opening the doors is actually optional since the attacker could decide to break the windows, and thus, doors-related attacks are optional (See the parenthesis around the (2) and (3) in the sequence constraint). To access to the system bus, the attacker can either connect with a remote interface (Bluetooth, Internet), either through a debug interface (OBD-II plug). For both solutions, he then needs to bypass the automotive system firewall.

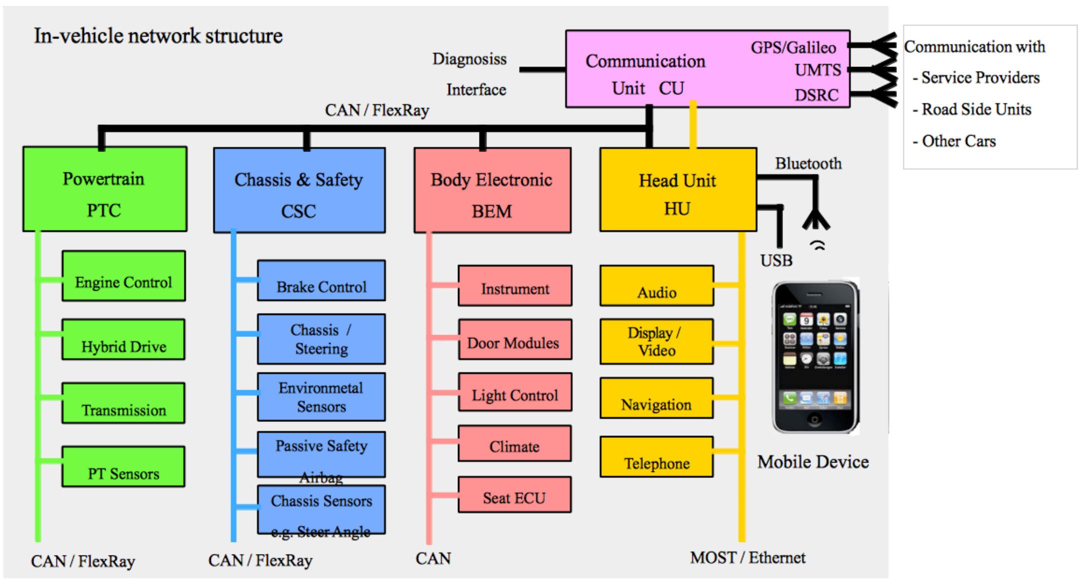
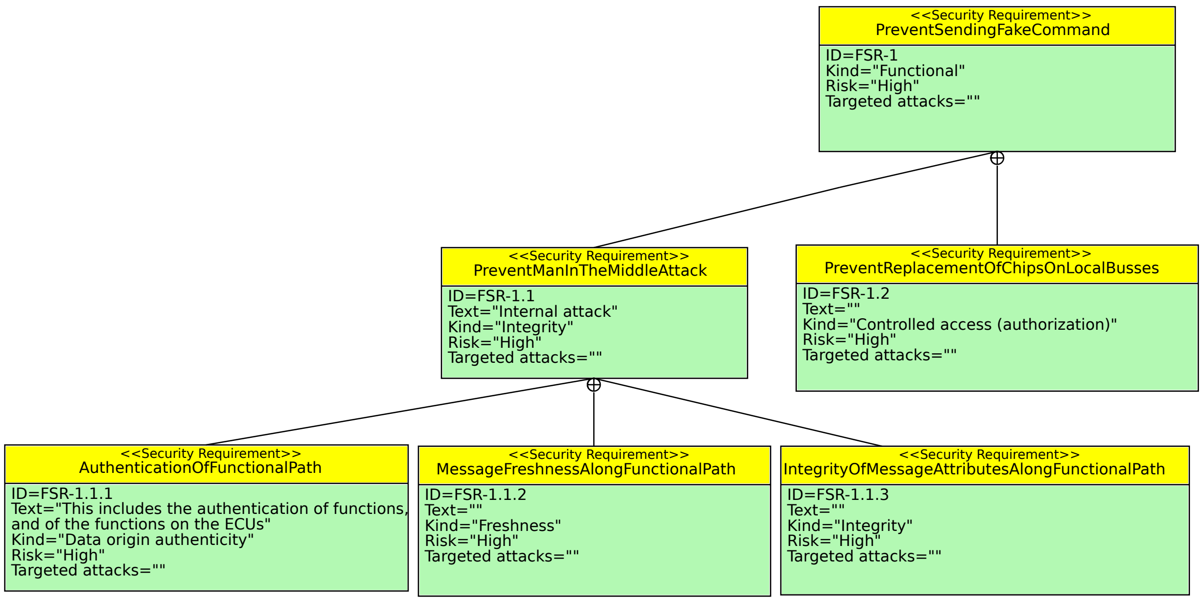


Figure 3: EVITA automotive reference architecture


<<Security Requirement>>
PreventSendingFakeCommand
ID=FSR-1
Kind="Functional"
Risk="High"
Targeted attacks=""

<<Security Requirement>>
PreventManInTheMiddleAttack
ID=FSR-1.1
Text="Internal attack"
Kind="Integrity"
Risk="High"
Targeted attacks=""

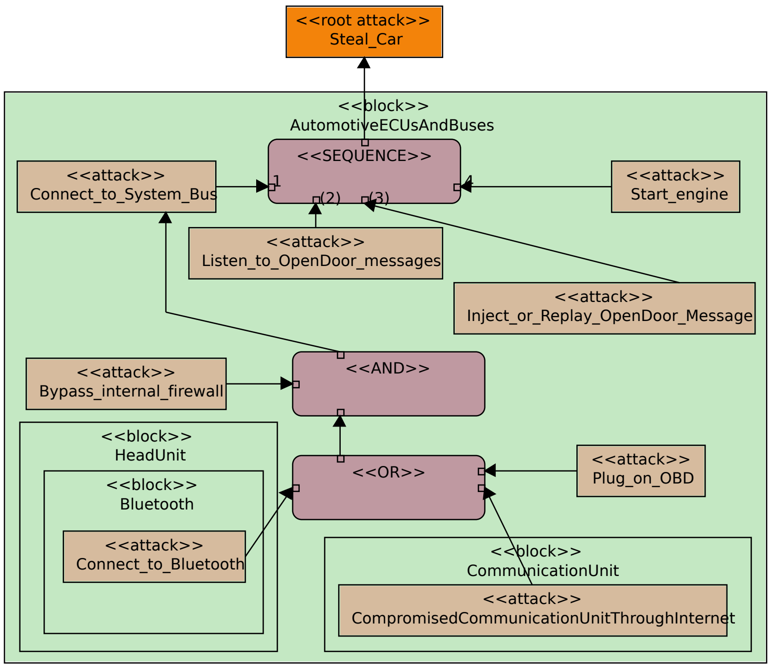
<<Security Requirement>>
PreventReplacementOfChipsOnLocalBusses
ID=FSR-1.2
Text=""
Kind="Controlled access (authorization)"
Risk="High"
Targeted attacks=""

<<Security Requirement>>
AuthenticationOfFunctionalPath
ID=FSR-1.1.1
Text="This includes the authentication of functions, and of the functions on the ECUs"
Kind="Data origin authenticity"
Risk="High"
Targeted attacks=""

<<Security Requirement>>
MessageFreshnessAlongFunctionalPath
ID=FSR-1.1.2
Text=""
Kind="Freshness"
Risk="High"
Targeted attacks=""

<<Security Requirement>>
IntegrityOfMessageAttributesAlongFunctionalPath
ID=FSR-1.1.3
Text=""
Kind="Integrity"
Risk="High"
Targeted attacks=""

Figure 4: Excerpt of security requirements



<<root attack>> Steal_Car
<<block>> AutomotiveECUsAndBuses
<<SEQUENCE>>
<<attack>> Connect_to_System_Bus
<<attack>> Start_engine
<<attack>> Listen_to_OpenDoor_messages
<<attack>> Inject_or_Replay_OpenDoor_Message
<<AND>>
<<attack>> Bypass_internal_firewall
<<block>> HeadUnit
<<block>> Bluetooth
<<attack>> Connect_to_Bluetooth
<<OR>>
<<attack>> Plug_on_OBD
<<block>> CommunicationUnit
<<attack>> CompromisedCommunicationUnitThroughInternet

Figure 5: Describing an attack with a SysML parametric diagram

6.3 Hardware/Software partitioning

A SysML block instance diagram is used to describe functions, and their relations. Data and event flows between functions can be described with ports and links. Semantically speaking, data flow do not model values of those data, but only the quantity of exchanged data: this grandly simplifies proofs and simulations at that level of abstraction (memo: validation takes into account both software and hardware components). Also, in the scope of complex systems, we have demonstrated that this does not impact partitioning decisions (Jaber, 2011).

The architectural and mapping models rely on a UML deployment diagram, but the allocation mechanism of SysML or MARTE could be used for that purpose. the mapping of functional blocks, and their communications, is based on artifacts added to deployment nodes, that latter being assets to be protected. They are likely to be also depicted in the attack diagrams.

In order to evaluate the impact of security mechanisms, two simulations are performed onto two different partitioning. A first simulation is performed on the automotive system with no security mechanisms The purpose of that simulation is to evaluate the CAN bus load, because it is in charge of conveying urgent and non-urgent messages between domains. The average load with no security is 40%.

A second simulation evaluates how the performance of the previous system is impacted by the key distribution protocol. For that purpose, a subset of the automotive system is modeled: cryptographic hardware accelerators (HSM - Hardware Security Module), the ECU asking for a session key (ECU1), the Key Master (KM), and the ECUs that will participate to the session and thus need the session key. The simulation shows a much higher load on the main CAN bus during a key distribution, and a much higher latency was observed for all classes of traffic not related to the key distribution (Schweppe et al., 2011): this security mechanism do impact the safety of the system. To solve that issue, one solution we experimented with was to split in several successive messages the authentication information.

7 CONCLUSION AND FUTURE WORK

Many attacks are now conducted on embedded systems and cyber-physical systems. A short time-to-market combined with strong safety and security requirements encourages the introduction of new development methodologies for those systems.

SysML-Sec integrates in the same development cycle semi-formal specifications of both safety and security features and properties. Simulations and formal proofs on models can be easily conducted with TTool, so as to assess architectural choices and design

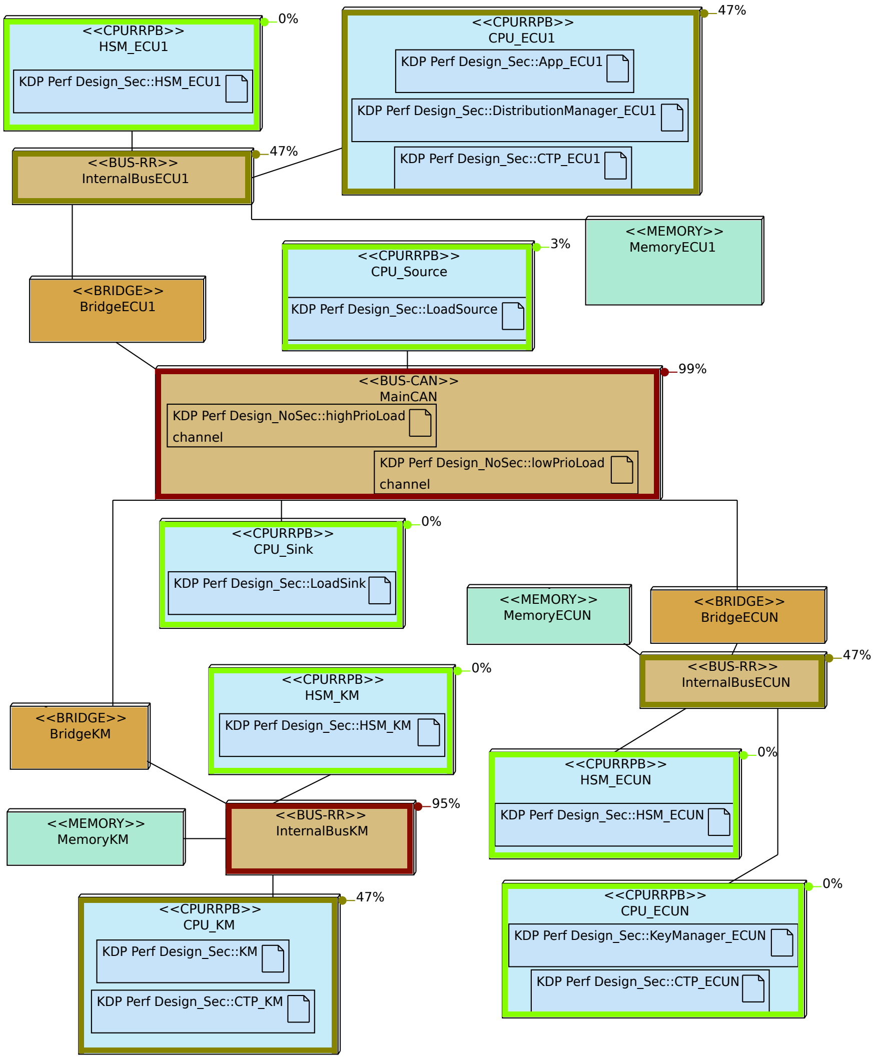


Figure 6: Simulation with the key distribution mechanism

choices, in terms of performance, safety properties, and security properties. Moreover, SysML is based on a well known and recognize language for system engineering, and is totally supported by TTool.

SysML-Sec has been defined and used in the scope of the EVITA project, that is, to secure an automotive embedded system. The case study presented in this paper is extracted from this project, and demonstrates the interest and choices of SysML-Sec.

One important objective of our work is now to add reasoning capabilities to SysML-Sec. More precisely, our goal is to verify that critical functionalities are not inhibited by the introduction of security mechanisms, e.g. message ciphering or network filtering. This could be done with logic inference rules.

REFERENCES

Apvrille, A. and Strazzere, T. (2012). Reducing the window of opportunity for android malware. gotta catch'em all. *Journal in Computer Virology*, 8(1-2):61–71.

Apvrille, L. and Becoulet, A. (2012). Prototyping an Embedded Automotive System from its UML/SysML Models. In *ERTSS'2012*, Toulouse, France.

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

Apvrille, L. and De Saqui-Sannes, P. (2013). Requirements analysis. *Embedded Systems: Analysis and Modeling with SysML, UML and AADL.*

Apvrille, L., Mifdaoui, A., and De Saqui-Sannes, P. (2010). Real-time distributed systems dimensioning and validation: The turtle method. *Studia Informatica Universalis*, 8(3):47–69.

Apvrille, L., Muhammad, W., Ameur-Boulifa, R., Coudert, S., and Pacalet, R. (2006). 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.

Apvrille, L. and Roudier, Y. (2013). SysML-Sec: A SysML environment for the design and development of secure embedded systems. In *APCOSEC 2013*, Yokohama, Japan.

Armando, A., Basin, D., Boichut, Y., Chevalier, Y., Compagna, L., Cuellar, J., Drielsma, P., Hem, P., Kouchnarenko, O., Mantovani, J., Mdersheim, S., Oheimb, D., Rusinowitch, M., Santiago, J., Turuani, M., Vigan, L., and Vigneron, L. (2005). The avispa tool for the automated validation of internet security protocols and applications. In Etessami, K. and Rajamani, S., editors, *Computer Aided Verification*, volume 3576 of *Lecture Notes in Computer Science*, pages 281–285. Springer Berlin Heidelberg.

Assolini, F. (2012). The Tale of One Thousand and One DSL Modems, kaspersky lab.

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

Beck, K. and Andres, C. (2004). *Extreme Programming Explained: Embrace Change (2nd Edition)*. Addison-Wesley Professional.

Bengtsson, J. and Yi., W. (2004). Timed automata: Semantics, algorithms and tools. In *Lecture Notes on Concurrency and Petri Nets*, pages 87–124. W. Reisig and G. Rozenberg (eds.), LNCS 3098, Springer-Verlag.

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

D. P. Eames, D. P. and Moffett, J. (1999). The integration of safety and security requirements. In *SAFECOMP*, pages 468–480.

Esser, S. (2011). Exploiting the iOS Kernel. In *BlackHat 2011.*

Garavel, H., Lang, F., Mateescu, R., and Serwe, W. (2007). CADP 2006: A Toolbox for the Construction and Analysis of Distributed Processes. In *Proceedings of the 19th International Conference on Computer Aided Verification CAV 2007.*

Huang, A. (2002). Keeping Secrets in Hardware: the Microsoft XBox Case Study, AI Memo 2002-008, Massachusetts Institute of Technology, Artificial Intelligence Laboratory. Technical report.

Jaber, C. (2011). *High-Level SoC Modeling and Performance Estimation Applied to a Multi-CoreImplementation of LTE EnodeB Physical Layer*. PhD thesis, Telecom ParisTech.

Kelling, E., Friedewald, M., Leimbach, T., Menzel, M., Säger, P., Seudié, H., and Weyl, B. (2009). Specification and Evaluation of e-Security Relevant Use cases. Technical Report Deliverable D2.1, EVITA Project.

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

Leboudec, J. and Thiran, P. (2001). *Network Calculus*. Springer Verlag LNCS volume 2050.

Maslennikov, D. (2010). Russian cybercriminals on the move: profiting from mobile malware. In *The 20th Virus Bulletin Internation Conference*, pages 84–89, Vancouver, Canada.

Maynor, D. (2006). Scada security and terrorism: We're not crying wolf! In *Invited presentation at BlackHat BH 2006. Presentation available at: https://www.blackhat.com/presentations/bh-federal-06/BH-Fed-06-Maynor-Graham-up.pdf*, USA.

Nhlabatsi, A., Nuseibeh, B., and Yu, Y. (2010). Security Requirements Engineering for Evolving Software Systems: a survey. Technical Report 1, The Open University.

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

OMG (2005). OMG Profile for Scheduling, Performance and Time. In *http://www.omg.org/spec/SPTP/.*

OMG (2012). OMG Systems Modeling Language. In *http://www.sysml.org/specs/*.

Pedroza, G. (2013). Assisting the design of secured applications for mobile vehicles. In *Ph.D. of Ecole doctorale informatique, télécommunications et électronique of Paris*.

Pietre-Cambacedes, L. and Bouissou, M. (2013). Cross-fertilization between safety and security engineering. *Rel. Eng. & Sys. Safety*, 110:110–126.

Proofpoint (2014). Your Fridge is Full of SPAM: Proof of An IoT-driven Attack. In *http://www.proofpoint.com/threatinsight/posts/your-fridge-is-full-of-spam-proof-of-a-Iot-driven-attack.php*.

Raspotnig, C. and Opdahl, A. L. (2013). Comparing risk identification techniques for safety and security requirements. *Journal of Systems and Software*, 86(4):1124–1151.

Schneier, B. (1999). Attack Trees: Modeling Security Threats.

Schweppe, H., Roudier, Y., Weyl, B., Apvrille, L., and Scheuermann, D. (2011). C2X communication: Securing the last meter. In *The 4th IEEE International Symposium on Wireless Vehicular Communications: WIVEC2011*, San Francisco, USA.

Teso, H. (2013). Aircraft Hacking. In *HITB Security Conference*, Amsterdam, The Netherlands.

Van Lamsweerde, A. (2007). Engineering Requirements for System Reliability and Security. *Software System Reliability and Security*, 9:196–238.

Vidal, J., de Lamotte, F., Gogniat, G., Soulard, P., and Diguet, J.-P. (2009). 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, 2009. DATE'09*, pages 226–231.

Waters, K. (2012). *All About Agile: Agile Management Made Easy!* CreateSpace Independent Publishing Platform.