Identification of authenticity requirements in systems of systems by functional security analysis

Andreas Fuchs and Roland Rieke
Fraunhofer Institute for Secure Information Technology (SIT)
Rheinstrasse 75, 64295 Darmstadt, Germany
Email: {andreas.fuchs,roland.rieke}@sit.fraunhofer.de

***Abstract*—Cooperating systems typically base decisions on information from their own components as well as on input from other systems. The reasoning process of one system that is part of a system of systems can take into account a wider scope that is out of reach of its own boundaries. Safety critical decisions of a system based on information from the environment, such as automatic emergency braking of a vehicle, raise severe concerns to security issues.**

In this paper we address the security engineering process for systems of systems. In particular we present a systematic and constructive approach to the authenticity requirements elicitation step in this process. The method is based on functional dependency analysis. It comprises the tracing down of functional dependencies over system boundaries right onto the origin of information. This spans a dependency graph with a safety critical function as root and the origins of decision relevant information as leaves.

Based on this graph, we deduce a set of authenticity requirements for the input from the leaves of the derivation graph. This set is comprehensive and defines the maximal set of authenticity requirements from the given functional dependencies. Furthermore, the proposed method avoids premature assumptions on the architectural structure and mechanisms to implement security measures.

I. Introduction

Architecting novel mobile systems of systems poses new challenges to getting the dependability and specifically the security requirements right as early as possible in the system design process. Security engineering is one important aspect of dependability [1]. This process addresses such issues as how to identify and mitigate risks resulting from connectivity and how to integrate security into a target architecture [2].

A typical application area for mobile systems of systems are vehicular communication systems in which vehicles and roadside units communicate in ad hoc manner to exchange information such as safety warnings and traffic information. As a cooperative approach, vehicular communication systems can be more effective in avoiding accidents and traffic congestion than current technologies where each vehicle tries to solve these problems individually. However, introducing dependence of possibly safety critical decisions in a vehicle from the information of the network raises severe concerns to security issues. Security is an enabling technology in this emerging field because without security those systems would not be possible at all. In some cases security is the main concern of the architecture [3].

The first step in the design of such a system is the requirements engineering process. This process typically covers at least the following three main activities [4], [5], [6]

1) preparative actions, such as a) the identification of the target of evaluation and the principal security goals and b) the elicitation of artifacts, e.g. use case and threat scenarios to support requirements definition as well as c) risk assessment.
2) the actual security requirements elicitation process
3) concluding actions, such as requirements categorisation and prioritisation, followed by requirements inspection

In this paper we address the security requirements elicitation step in this process. We present a model-based approach to systematically identify security requirements for systems to be designed in a systems of systems context. Our contribution comprises the following distinctive features:

- *Identification of a consistent and complete set of authenticity requirements.* Security requirements need to be explicit, precise, adequate, non-conflicting with other requirements and complete [7].
  We show how the overall security goal of maximum authenticity of all information from cooperating entities leads to a comprehensive set of authenticity requirements. Once an exhaustive list of requirements is identified, a requirements categorisation and prioritisation process can evaluate them according to a maximum acceptable risk strategy.
- *Security mechanism independence.* The most common problem with security requirements, when they are specified at all, is that they tend to be accidentally replaced with security-specific architectural constraints that may unnecessarily constrain the security team from using the most appropriate security mechanisms for meeting the true underlying security requirements [8].
  In our approach we avoid to break down the overall requirements to requirements for the edges of the functional dependency graph prematurely. So the requirements identified by this approach are independent of decisions not only on concrete security enforcement mechanisms to use, but also on the structure, such as hop-by-hop versus end-to-end security measures.

Throughout this paper we use the following terminology taken from [1]: A *system* is an entity that interacts with other entities, i.e., other systems. These other systems are the *environment* of the given system. A *system boundary* is the common frontier between the system and its environment. Such a system itself is composed of *components*, where each component is yet another system. Furthermore, in [1] the *dependence* of system A on system B represents the extent to which system A's dependability is affected by that of system B. This work though focuses on purely functional aspects of dependence and omits quantitative reasoning.

For the approach proposed, we describe the *function* of such a system by a *functional model* and treat the components as atomic and thus we do not make preliminary assumptions regarding their inner structure. Rather, the adaption to a concrete architecture is considered to be a task within a follow-up refinement and engineering process.

II. Related Work

The development of new security relevant systems that interact to build new systems of systems requires the integration of a security engineering process in the earliest stages of the development lifecycle. This is specifically important in the development of systems where security is the enabling technology that makes new applications possible.

A comprehensive concept for an overall security requirements engineering process is described in detail in [5]. The authors propose a 9 step approach called SQUARE (Security Quality Engineering Methodology). The elicitation of the security requirements is one important step in the SQUARE process. In [6] several concrete methods to carry out this step are compared. These methods are based on misuse cases (MC), soft systems methodology (SSM), quality function deployment (QFD), controlled requirements expression (CORE), issue-based information systems (IBIS), joint application development (JAD), feature-oriented domain analysis (FODA), critical discourse analysis (CDA) as well as accelerated requirements method (ARM). A comparative rating based on 9 different criteria is also given but none of these criteria measures the completeness of the security requirements elicited by the different methods.

A similar approach based on the integration of Common Criteria (ISO/IEC 15408) called SREP (Security Requirements Engineering Process) is described in [4]. However the concrete techniques that carry out the security requirements elicitation process are given only very broadly. A threat driven method is proposed but is not described in detail.

In [8] different kinds of security requirements are identified and informal guidelines are listed that have proven useful when eliciting concrete security requirements. The author emphasises that there has to be a clear distinction between security requirements and security mechanisms.

In [9] it is proposed to use Jackson‘s problem diagrams to determine security requirements which are given as constraints on functional requirements. Though this approach presents a methodology to derive security requirements from security goals, it does not explain the actual refinements process, which leaves open, the degree of coverage of requirements, depending only on expert knowledge.

In [10] actor dependency analysis is used to identify attackers and potential threats in order to identify security requirements. The so called i^* approach facilitates the analysis of security requirements within the social context of relevant actors. In [11] a formal framework is presented for modelling and analysis of security and trust requirements at an organisational level. Both of these approaches target organisational relations among agents rather than functional dependence. Those approaches might be utilised complementary to the presented. Also the output of organisational relations analysis may be an input to our functional security analysis.

In [7] anti-goals derived from negated security goals are used to systematically construct threat trees by refinement of these anti-goals. Security requirements are then obtained as countermeasures. This method aims to produce more complete requirements than other methods based on misuse cases. The refinement steps in this method can be performed informally or formally.

III. Motivation

The derivation of security requirements in general, especially the derivation of authenticity requirements represents an essential building block for system design. With an increase in the severity of safety-relevant systems' failures the demand increases for a systematic approach of requirements derivation with a maximum coverage. Also during the derivation of security requirements, no pre-assumptions should be made about possible implementations.

We will further motivate this with respect to the requirements derivation process with an example from the field of vehicle-to-vehicle communications and demonstrate the common mistakes.

A. Example Use Case

For a better illustration of the described problems we will refer to an example, illustrating use case description for a possible vehicle-to-vehicle scenario:

1) Use Case 1: A vehicle's Electronic Stability Protection (ESP) sensor recognises that the ground is very slippy when accelerating in combination with a low temperature. In order to warn successive vehicles about a possibly icy road, the vehicle sends out information about this danger including the GPS position data, where the danger was detected.

2) Use Case 2: A vehicle receives a warning about an icy road at a certain position. It compares the information to its own position and heading and signals the driver a warning, if the dangerous area lies up front. Additionally the vehicle will retransmit the warning, given that the position of this occurrence is not too far away.

B. Common Approaches

There are several possible approaches, that may be taken, depending on the system architect's background.

An architect with a background in Mobile Adhoc Networks (MANETs) would first define the data origin authentication

[12] of the transmitted message. In a next step he may reason about the trustworthiness of the transmitting system.

An architect with a background in Trusted Computing [13] would first require for the transmitting vehicle to attest for its behaviour [14]. Advanced experts may use the techniques of sealing, binding, key restrictions and TPM-CertifyKey to validate the trustworthiness and bind the transmitted data to this key [15].

A distributed software architect may first start to define the trust zones. This would imply that some computational means of composing slippy wheels with temperature and position happen in an untrusted domain. Results may be the timestamped signing of the sensor data and a composition of these data at the receiving vehicle.

C. Problem Evaluation

The presented methods shall only illustrate a few different approaches that might be taken, when challenged with the development of secure architecture like this. Still, one can see, that very different types of security requirements are the outcome. Some of which leave attack vectors open, such as the manipulation of the sending or receiving vehicles internal communication and computation.

Another conclusion that can be derived from these examples is related to premature assumption about the implementation. Whilst in one case the vehicle is seen as a single computational unit that can be trusted, in another case it has to attest for its behaviour when sending out warnings. The third analysis of the same use cases however, requires for a direct communication link and cryptography between the sensors and the receiving vehicle and the composition of data is moved to the receiver side.

Though all of the approaches may lead to the same level of security for the designed architecture, there is no means by which they can be compared regarding the security requirements, that they fulfil. By analysis of the authors, this is a direct result of falsely defined system boundaries, where security requirements are formulated against internal subsystems rather than the system at stake itself. The choice of the appropriate abstraction level and system boundaries constitutes a rather big challenge to systems of systems design, especially in a system of systems application like the one presented here.

IV. Approach

The approach described in the following can be decomposed into three basic steps. The first one is the derivation of the functional model from the use case descriptions in terms of an action oriented system. In a second step the system at stake is defined and possible instantiations of the first functional model are elaborated. In a third and final step, the actual requirements are derived in a systematic way, resulting in a consistent and complete set of security requirements.

A. Functional Model

For the description of the functional model from the use cases an action-oriented approach is chosen. The approach is based on the work from [16]. For reasons of simplicity and readability the formal description of the model is omitted here and a graphical representation is used to illustrate the behaviour of the evaluation target.

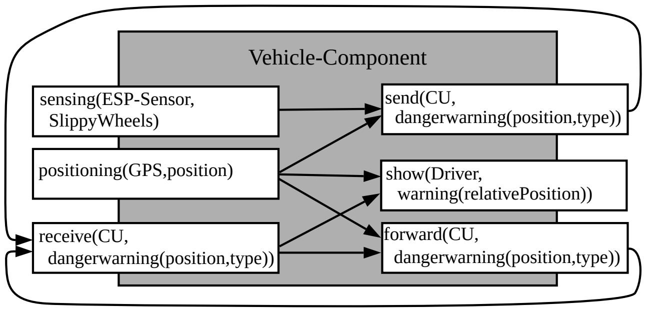

Vehicle-Component
sensing(ESP-Sensor, SlippyWheels)
positioning(GPS,position)
receive(CU, dangerwarning(position,type))
send(CU, dangerwarning(position,type))
show(Driver, warning(relativePosition))
forward(CU, dangerwarning(position,type))

Fig. 1. Example Functional Component Model

A functional model can be derived from a use case description by identifying the atomic actions in the use case description. These actions are set in relation by defining the functional flow among them. The functional flow considers Input/Output operations as well as interprocess communication, or even interthread communication. The important factor is, that some kind of data transport or control flow takes place. Therefore, it does not matter, if the data is transmitted by an action triggering push-operations, or by pull-operations. However, if availability and lifeness analysis were taken into account as well, the analysis of control flow would need to be addressed as well.

In the case of highly distributed systems and especially a distributed system of distributed systems, it is very common, that use cases do not cover a complete functional cycle throughout the whole system under investigation. Rather only certain components of the system are described regarding their behaviour. This must be kept in mind, when deriving the functional model. In order to clarify this distinction, functional models that describe only parts of the overall system behaviour will be called *functional component model*.

Example - Functional Component Model: Regarding the example use cases given in Section III-A the resulting functional component model for a vehicle can be illustrated as shown in Figure 1. In this context, functional flow arrows outside of the vehicle's boundaries do refer to functional flows between different instances of the component, whilst internal flow arrows refer to flows within the same instance of the component. For the given example, the external flows represent data transmission of one vehicle to another, whilst the internal flows represent communication within a single vehicle.

B. System Instances

Based on the functional component model, one may now start to reason about the overall system. The synthesis of the inner and the outer system behaviour builds the global system behaviour. The instantiation of the component described in the use cases has to be done with care though. It is not the goal to instantiate every possible combination of e.g. names, as those system instances would be isomorphic. Rather, it is desired to construct all *structurally different* possibilities that may be used to construct a system.

Finally, all possible instances may be regrouped and the system's boundary actions (denoting the actions that are triggered by or influence the system environment) have to be identified. These will be the basis for the security requirements definition in the next step.

Example - System Instances: In Figure 2 an example for possible instances of the vehicle description in a distributed vehicle-to-vehicle scenario is presented. Note that the number of instance for such a scenario seems to be infinite in this representation, though, of course, the number is firstly restricted to the number of vehicles on earth. Secondly, the forwarding of a message was restricted to the distance from the danger, that is being warned about. We could therefore assume a certain (unclear) number, but the security requirements should be general enough, to cover all these cases, e.g. by utilising a description in a parameterised way.

C. Functional Security Requirement Identification

The set of possible instantiations of the functional component model is used in a next step to derive security requirements. First, the boundary actions of the system model instances have to be determined. The term *boundary action* in this sense refers to the actions that form the interaction of the internals of the system with the outside world. These are actions that are either triggered by occurrences outside of the system or actions that involve changes to the outside of the system.

With the boundary actions being identified, one may now follow the functional graph backwards. Beginning with the boundary actions by which the system takes influence on the outside, we may propagate backwards along the functional flow. These backwards references basically describe the functional dependencies of actions among each other. From the functional dependency graph we may now identify the end points - the boundary actions that trigger the system behaviour that depends on them. Between these and the corresponding starting points, a requirement exists, that without such an action happening as input to the system, the corresponding output action must not happen as well. From this we formulate the security goal of the system at stake:

Whenever a certain output action happens, the input action that presumably led to it, must actually have happened.

This requirement shall now be enriched by additional parameters. In particular, it shall be identified which is the entity that must be assured of the aforementioned requirement. With these additional parameters set, we may utilise the definition of authenticity from the formal framework of Fraunhofer SIT [17], to specify the identified requirements.

The syntax used to describe these requirements in parameterised form is defined as follows:

Definition 1: $Authentic(A, B, P)$: Whenever an action B happens, then it must be authentic for an Agent P, that in any course of events that seem possible to him, a certain action A has happened (for a formal definition we refer the reader to [17]).

It shall be noted, that the requirements process in this case utilised positive formulations of how the system should behave, rather than preventing a certain malicious behaviour. Also it has to be stressed, that this approach guarantees for the system / component architect to be free regarding the choice of concepts during the security engineering process.

Special care has to be taken though in cases of unlimited possible system instances - as is the case with the presented example. In practice, a certain boundary may usually be drawn, e.g. the maximum number of computers is naturally restricted. Therefore, there always exists only a countable and limited number of instances of the system. Accordingly we may define sets of requirements regarding certain large and undefined but limited sets of instances.

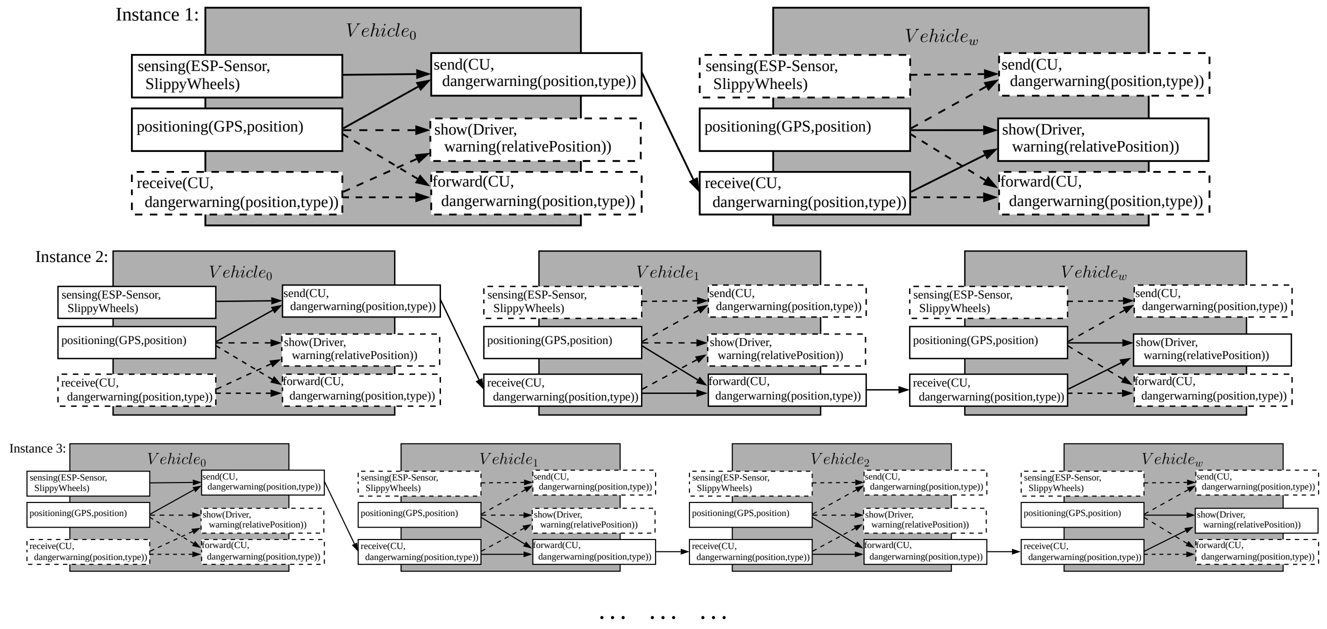
Finally, requirements (especially those referring to large sets) may be evaluated regarding to their importance to the system. This manual analysis may reveal that certain functional dependencies are presented only for performance reasons. This can be valuable input for the architects as well, and sometimes reveal premature decisions about mechanisms, that were already done during the use case definition phase.

Though it might appear possible that this approach may form infinite circles among the system actions, this cannot happen for well-defined use cases. This actually originates from the fact that every action represents a progress in time. Accordingly an infinite loop among actions in the system would indicate that the system described will not terminate.

The requirements derivation process will however highlight every functional dependency that is described within the use cases. Accordingly, when the use case description incorporates more than the sheer safety related functional description, additional requirements may arise. Therefore, the requirements have to be evaluated towards their meaning for the system's safety. Whilst one can be assured not to have missed any safety relevant requirement, this is a critical task, that should be performed with care, in order not to misjudge a requirement's relevance and thereby induce security holes.

D. Formalisation

Formally, the functional flow among actions can be interpreted as an ordering relation ζ_i on the set of actions Σ_i in a certain system instance i. To derive the requirements the reflexive transitive closure ζ_i^* is constructed. By construction rule, the functional flow graph is sequential and free of loops, as every action can only depend on past actions. Accordingly, the relation is anti-symmetric. ζ_i^* is a partial order on Σ_i, with the maximal elements max corresponding to the outgoing boundary actions and the minimal elements min corresponding to the incoming boundary actions. After restricting ζ_i^* to these elements $\chi_i = \{(x, y) \in \Sigma_i \times \Sigma_i \mid (x, y) \in \zeta_i^* \wedge x \in min \wedge y \in max\}$ this new relation represents the authenticity requirements for the corresponding system instance: *For all* $x, y \in \Sigma_i$ *with* $(x, y) \in \chi_i$: $auth(x, y, stakeholder(y))$ *is a requirement.* Accordingly the union of all these requirements



Instance 1:
$Vehicle_0$
$Vehicle_w$
sensing(ESP-Sensor, SlippyWheels)
positioning(GPS,position)
receive(CU, dangerwarning(position,type))
send(CU, dangerwarning(position,type))
show(Driver, warning(relativePosition))
forward(CU, dangerwarning(position,type))
Instance 2:
$Vehicle_1$
Instance 3:
$Vehicle_2$

Fig. 2. Example Functional Model Instances

for the different instances pose the set of requirements for the whole system. However, a lot of correlations between the system instances should appear, such that either requirements overlap, or first-order predicates can express them.

Example - Security Requirements of Authenticity: For the given system model instances, we may now identify the authenticity requirements for the action $show(V_w, D_w, warn(rP))$ (with V=Vehicle, D=Driver, CU=Communication Unit, ESP=Electronic Stability Protection sensor, $warn$=warning, pos=positioning, $sens$=sensing, rP=relativePosition, pD=positionData, sW=slippyWheels, dw=dangerwarning). Graphically, this could be done by reversing the arrows and removing the dotted arrows and boxes.
Formally, for the first system instance, we can analyse:

$$\begin{aligned}\zeta_1 = \{&(sens(ESP(V_0), sW), send(CU(V_0), dw)),\\ &(pos(GPS(V_0), pD), send(CU(V_0), dw)),\\ &(send(CU(V_0), dw), rec(CU(V_w), dw)),\\ &(pos(GPS(V_w), pd), show(V_w, D, warn(rP))),\\ &(rec(CU(V_w), dw), show(V_w, D, warn(rP)))\}\end{aligned}$$

$$\begin{aligned}\zeta_1^* = \zeta_1 \cup \{&(x,x)|x \in \Sigma\} \cup \{\\ &(sens(ESP(V_0), sW), rec(CU(V_w), dw)),\\ &(sens(ESP(V_0), sW), show(V_w, D_w, warn(rP))),\\ &(pos(GPS(V_0), pD), rec(CU(V_w), dw)),\\ &(pos(GPS(V_0), pD), show(V_w, D_w, warn(rP)))\}\end{aligned}$$

$$\begin{aligned}\chi_1 = \{&(sens(ESP(V_0), sW), show(V_w, D_w, warn(rP))),\\ &(pos(GPS(V_0), pD), show(V_w, D_w, warn(rP))),\\ &(pos(GPS(V_w), pD), show(V_w, D_w, warn(rP)))\}\end{aligned}$$

An analysis for the second system instance will result in:

$$\begin{aligned}\chi_2 = \{&(sens(ESP(V_0), sW), show(V_w, D_w, warn(rP))),\\ &(pos(GPS(V_0), pD), show(V_w, D_w, warn(rP))),\\ &(pos(GPS(V_w), pD), show(V_w, D_w, warn(rP))),\\ &(pos(GPS(V_1), pD), show(V_w, D_w, warn(rP)))\}\end{aligned}$$

And the third system instance will result in:

$$\begin{aligned}\chi_3 = \{&(sens(ESP(V_0), sW), show(V_w, D_w, warn(rP))),\\ &(pos(GPS(V_0), pD), show(V_w, D_w, warn(rP))),\\ &(pos(GPS(V_w), pD), show(V_w, D_w, warn(rP))),\\ &(pos(GPS(V_1), pD), show(V_w, D_w, warn(rP))),\\ &(pos(GPS(V_2), pD), show(V_w, D_w, warn(rP)))\}\end{aligned}$$

The first three elements in each χ_i will obviously always be the same. The rest of the elements can be expressed in terms of first-order predicates. This leads to the following authenticity requirements for all possible system instances:

1) $auth(\ pos(GPS(V_w), pD),$
$\quad show(V_w, D_w, warn(rP)), D_w\ )$
2) $auth(\ pos(GPS(V_0), pD),$
$\quad show(V_w, D_w, warn(rP)), D_w\ )$
3) $auth(\ sens(ESP(V_0), sW),$
$\quad show(V_w, D_w, warn(rP)), D_w\ )$
4) $\forall\ V_x \in V_{forward} : auth(\ pos(GPS(V_x), pD),$
$\quad show(V_w, D_w, warn(rP)), D_w\ )$

As mentioned above, the resulting requirements have to be evaluated regarding their meaning for the functional safety of the system. For the first three requirements the argumentation is very straight forward regarding, why they have to be fulfilled:

1) It must be authentic for the driver, that the relative position of the danger he/she is warned about is based on correct position information of his/her vehicle.
2) It must be authentic for the driver, that the position of the danger he/she is warned about is based on correct position information of the vehicle issuing the warning.
3) It must be authentic for the driver, that the danger he/she is warned about is based on correct sensor data.

The last requirement 4) however must be further evaluated. Studying the use case, we see, that this functional dependency originates from the geographic based forwarding policy. This policy is introduced for performance reasons, such that bandwidth is saved by not flooding the whole network. Braking this requirement would therefore result either in a smaller or larger broadcasting area. As bad as those cases may be, they cannot cause the warning of a driver, that should not be warned. Therefore we do not consider requirement 4 to be a safety related authenticity requirement. It can be considered a requirement regarding availability by preventing the denial of a service or unintended consumption of bandwidth.

E. Further steps

Starting from this set of very high-level requirements, the security engineering process may start. This will include decisions regarding the mechanisms to be included. Accordingly the requirements may be refined to more concrete requirements in this process. The design and refinement process may reveal further requirements regarding the internals of the system, that have to be addressed as well.

V. Conclusion

The presented approach for deriving safety critical authenticity requirements in systems of systems solves several issues compared to existing approaches. It incorporates a clear scheme that will ensure a consistent and complete set of security requirements. Also it is based directly on the functional analysis, ensuring the safety of the system at stake. The systematic approach that incorporates formal semantics leads directly to the formal validation of security, as it is required by certain evaluation assurance levels of Common Criteria (ISO/IEC 15408). Furthermore the difficulties of designing systems of systems are specifically targeted.

In practice, the method described here is applied in the project EVITA [1] to derive authenticity requirements for the development of a new automotive on-board architecture utilising vehicle-to-vehicle and vehicle-to-infrastructure communication.

A. Future Work

Future work may include the derivation of confidentiality requirements in a similar way, as was presented here. Though this will require for different security goals, as confidentiality is not related to safety in a similar way, but rather privacy. Non-Repudiation may also be a target, that should be approached in cooperation with lawyers, in order to find the relevant security goals. Furthermore, the refinement throughout the design process should be evaluated regarding possibility of formalising it in schemes with respect to the security requirements refinement process.

Acknowledgment

Part of the work presented in this paper was developed within the project EVITA being co-funded by the European Commission within the Seventh Framework Programme.

[1]EVITA (E-Safety Vehicle Intrusion Protected Applications) http://www.evita-project.org

References

[1] A. Avizienis, J.-C. Laprie, B. Randell, and C. E. Landwehr, “Basic concepts and taxonomy of dependable and secure computing,” *IEEE Trans. Dependable Sec. Comput.*, vol. 1, no. 1, pp. 11–33, 2004.

[2] D. J. Bodeau, “System-of-Systems Security Engineering,” in *In Proc. of the 10th Annual Computer Security Applications Conference, Orlando, Florida*. IEEE Computer Society, 1994, pp. 228–235.

[3] P. Papadimitratos, L. Buttyan, J.-P. Hubaux, F. Kargl, A. Kung, and M. Raya, “Architecture for Secure and Private Vehicular Communications,” in *IEEE International Conference on ITS Telecommunications (ITST)*, Sophia Antipolis, France, June 2007, pp. 1–6.

[4] D. Mellado, E. Fernández-Medina, and M. Piattini, “A common criteria based security requirements engineering process for the development of secure information systems,” *Comput. Stand. Interfaces*, vol. 29, no. 2, pp. 244–253, 2007.

[5] N. R. Mead and E. D. Hough, “Security requirements engineering for software systems: Case studies in support of software engineering education,” in *CSEET ’06: Proceedings of the 19th Conference on Software Engineering Education & Training*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 149–158.

[6] N. R. Mead, “How To Compare the Security Quality Requirements Engineering (SQUARE) Method with Other Methods ,” Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU/SEI-2007-TN-021, 2007.

[7] A. van Lamsweerde, “Elaborating security requirements by construction of intentional anti-models,” in *ICSE ’04: Proceedings of the 26th International Conference on Software Engineering*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 148–157.

[8] D. Firesmith, “Engineering security requirements,” *Journal of Object Technology*, vol. 2, no. 1, pp. 53–68, 2003.

[9] C. B. Haley, R. C. Laney, J. D. Moffett, and B. Nuseibeh, “Security requirements engineering: A framework for representation and analysis,” *IEEE Trans. Software Eng.*, vol. 34, no. 1, pp. 133–153, 2008.

[10] L. Liu, E. Yu, and J. Mylopoulos, “Analyzing security requirements as relationships among strategic actors,” in *2nd Symposium on Requirements Engineering for Information Security (SREIS’02)*, 2002.

[11] P. Giorgini, F. Massacci, J. Mylopoulos, and N. Zannone, “Requirements engineering meets trust management: Model, methodology, and reasoning,” in *In Proc. of iTrust 04, LNCS 2995*. Springer-Verlag, 2004, pp. 176–190.

[12] R. Shirey, “Internet Security Glossary, Version 2,” RFC 4949 (Informational), Aug. 2007. [Online]. Available: http://www.ietf.org/rfc/rfc4949.txt

[13] T. C. Group, “TCG TPM Specification 1.2 revision 103,” www.trustedcomputing.org, 2006.

[14] R. Sailer, X. Zhang, T. Jaeger, and L. van Doorn, “Design and implementation of a TCG-based integrity measurement architecture,” in *Proceedings of the 13th USENIX Security Symposium*. USENIX Association, 2004.

[15] A.-R. Sadeghi and C. Stüble, “Property-based attestation for computing platforms: caring about properties, not mechanisms,” in *NSPW ’04: Proceedings of the 2004 workshop on New security paradigms*. New York, NY, USA: ACM, 2004, pp. 67–77.

[16] P. Ochsenschläger, J. Repp, and R. Rieke, “Abstraction and composition – a verification method for co-operating systems,” *Journal of Experimental and Theoretical Artificial Intelligence*, vol. 12, pp. 447–459, June 2000, copyright: ©2000, American Association for Artificial Intelligence (www.aaai.org). All rights reserved. [Online]. Available: http://sit.sit.fraunhofer.de/smv/publications/download/flairs-2000c.pdf

[17] S. Gürgens, P. Ochsenschläger, and C. Rudolph, “Authenticity and provability - a formal framework,” in *Infrastructure Security Conference InfraSec 2002*, ser. Lecture Notes in Computer Science, vol. 2437. Springer Verlag, 2002, pp. 227–245.