Exploiting Functional Models to Assess the Security Aspect in Embedded System Design

Ingo Stierand[1] and Sunil Malipatlolla[2]

[1] Carl von Ossietzky Universität Oldenburg,
26111 Oldenburg, Germany
stierand@informatik.uni-oldenburg.de
[2] OFFIS - Institute for Information Technology,
26121 Oldenburg, Germany
sunil.malipatlolla@offis.de

Abstract. Conventionally, automotive embedded systems are assessed for evaluating various different aspects such as safety, functionality, and real-time. However, the inclusion of security aspect, which indeed is becoming increasingly important in modern day cars, has a significant impact on the above aspects, especially on functionality and real-time. This impact would be clearly visible in the functional model of the embedded system because including security features modifies the data flow in the system. Thus, the goal of this contribution is to assess and evaluate the security aspect in such systems by exploiting their functional models. Such an assessment further results in establishing a possible relation between real-time formal analysis and the existing security theory. For this, a formal approach well-known from real-time embedded domain is utilized in here.

Keywords: Real-Time, Embedded System, Formalization, Security Protocols, Validation.

1 Introduction

With an increased inclusion of electronics in automotive systems, they are becoming more and more vulnerable to attacks such as manipulation of data packets and malicious system updates. Thus, ensuring the security of such systems is a crucial task, particularly in safety relevant systems, where unintended modifications can lead to malfunctioning of a system. To achieve this, good methods are required to evaluate the security of a system. On the other hand, such an evaluation should not be isolated but must be done in an integrated manner. This means, though the system is initially modeled for functional and real-time aspect evaluation, a possible direction for evaluating the security aspect must further be given utilizing the same model.

In the above context, an automotive embedded sub-system, as depicted in Figure 1, is considered as the target system here. The system comprises of a sensor, an actuator, an Electronic Control Unit (ECU) with a processor and a

Sabu M. Thampi et al. (Eds.): SSCC 2013, CCIS 377, pp. 90–97, 2013.

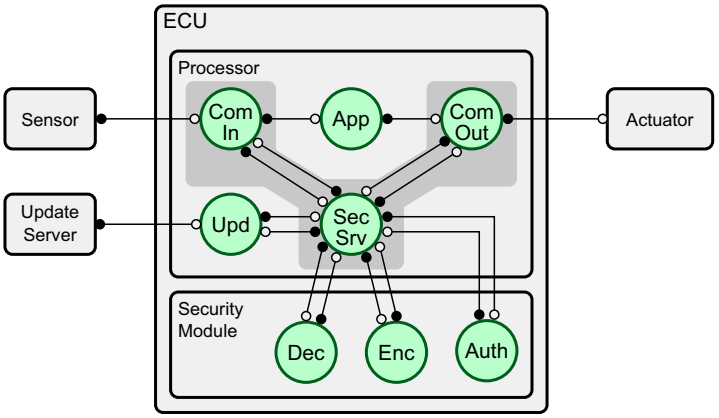

Fig. 1. An Example Embedded Sub-System with an ECU

security module, and an update server. It is a real-time system which has to satisfy its real-time properties, such as meeting deadlines, in addition to guaranteeing the functional correctness. The system realizes a simple real-time control application, where sensor data are processed by the control application in order to operate the plant due to an actuator. Though the concrete control application is not of interest in here, it might represent the engine control of a car or a driver assistant system such as an Automatic Breaking System (ABS).

In specific, Figure 1 depicts the functional model i.e., the control functionality performed by the `App` task and the update functionality executed by the `Upd` task. Encrypted data coming from the `Sensor` is decrypted through `SecSrv` utilizing the `Dec` function before feeding it to the task `App`. After performing the control operation, the output is again encrypted through `SecSrv` utilizing `Enc` block before forwarding it to the `Actuator`. Similarly, the update data from the `Update Server` is authenticated through `SecSrv` utilizing `Auth` block before loading into the system.

It can be seen that including the security module has a deep impact on the functional model of the system and thus the data flow in it. For example, due to the addition of authentication block for securing the system against malicious updates from the update server modifies the path of the data flow in the system. This is clearly visible in Figure 1, i.e., without an authentication block the data from the update server would have been directly loaded into the system by the `Upd` task instead of traversing through the tasks `Upd`, `SecSrv`, and the `Auth` block. Further, such a modification in the data flow due to the inclusion of security module has an impact on the real-time aspect of the system, as shown in our previous work [12].

On the other hand, such a functional model provides the capability for the system designer to model the possible attack points in its data flow path as detailed below. For example, an attacker may try to intercept the data from the

`Sensor` and clone it later if it is not encrypted or he may deceive the system for being the `UpdateServer` to load malicious update data into the system. Given this, the possible attack points are the communication channels between `Sensor` and ECU, and `UpdateServer` and ECU, respectively. Thus, the goal of this contribution is to assess the security aspect of such a system by exploiting its functional model and the corresponding data flow. For this, a formal approach, which is well-known for modeling the real-time systems, is utilized in here [3].

The rest of the paper is organized as follows. Section 2 gives a brief description of the existing work in the literature. Section 3 illustrates the proposed approach for assessing the security aspect utilizing the system functional model. We show that our functional models can be used as the input for security analysis, while ensuring that the semantics of both models are preserved. Section 4 sketches further steps towards integration by showing how the functional model can be exploited for confining the target security model. Section 5 concludes the paper and gives some hints on future work.

2 Related Work

Real-time task networks are well-established formalisms at the considered phase of system design, where the system functionality is mapped to a particular hardware architecture, including scheduling policies provided by the operating system(s). The relation between real-time systems and data flow models have been intensively studied over the last decade [2,10]. On the other hand, various security models are based on the data flow paradigm, such as the one from [13], which is mainly considered in this work. To the best of our knowledge, similar considerations for exploiting real-time formalisms in the security domain are yet to be made.

The authors Yip et al. in [15] proposed a new language run-time, referred to as *RESIN*, that helps to prevent security vulnerabilities, by allowing programmers to specify application-level data flow assertions. For this, it utilizes the mechanisms of policy objects associated with data, data tracking as data flows through an application, and filter objects that define data flow boundaries and control data movement. Though the goal of this work is to protect the software applications such as PHP and python against the existing vulnerabilities, our goal is to assess the security aspect based on data flow model in an embedded system.

The work in [6] proposes a generic framework for evaluating whether given information flows can cause security issues. The objective of the approach is to ensure that data objects do not leave the security class they are assigned to, or transitions into another class that is allowed by the information flow ($\rightarrow$) operator. The approach defines flows based on sequences of functions on the data objects. There also exist various well-established formalisms such as CSP [11] and Petri nets [5] that have been exploited for security analysis.

However, unlike the formalism utilized in here, none of the above approaches were originally developed for an architectural design.

3 Approach

3.1 Functional Real-Time Model

Real-time scheduling analysis [14] is an important building block in design processes for safety relevant systems. While early design phases consider the target hardware in a rather abstract way, if any, real-time scheduling analysis considers applications when they are deployed onto hardware with processors, buses and other components. Maybe the most popular formalisms to model real-time systems are so-called *task networks*, where tasks represent the individual software elements of the system. The tasks are connected in a graph structure, where the connections between tasks denote execution precedences. Typically, a real-time model also consists of a simple notion of hardware architectures, representing processing elements and buses in the case of distributed architectures. The tasks are allocated to the respective processing elements, representing their execution on the architecture elements.

In the following we introduce a particular class of task network models that allows to characterize individual data flows in the systems. The formalism actually is a simplified version of the one discussed in [3]. Each task is equipped with a set of input and output ports. The data flowing into and out of a task is modeled by a set of events that may occur at the respective ports. Tasks are connected by channels. A channel characterizes the flow of events between the individual tasks.

Definition 1. *A task network is a tuple* $N = (\Sigma, P, E, T, C)$ *where:*

- Σ *is a set of events,*
- P *is a set of ports,*
- $E : \Sigma \to P$ *induces for each port* $p \in P$ *a set* $\Sigma(p) \subseteq \Sigma$ *of events,*
- T *is a set of software tasks, where* $t \in T$ *is a tuple* (P_i, ψ, P_o)*:*
  - $P_i, P_o \subseteq P$ *are the input and output ports of* t*,*
  - $\psi : \bigcup_{p \in P_i} \Sigma(p) \to \Sigma$ *is the execution function of the task. We require that* $\psi(\sigma) \in \bigcup_{p \in P_o} \Sigma(p)$*.*
- $C \subseteq P \times P$ *is a set of channels.* ◇

Note that the definition omits timing annotations, with which a task network can be checked whether it satisfies given timing requirements, as for example discussed in previous work [12].

The execution function ψ of a task defines the reaction of the task with respect to its activation by an incoming data object on its input port(s). Together with our notion of connectivity by channels, the formalism allows to model a wide range of (deterministic) data flows in a system. This is depicted for example in Figure 2, which illustrates the update service scenario of our initial system model (c.f. Figure 1) in more detail. It is assumed that the update service receives two different kinds of data packets, namely p and p'. A p packet denotes a packet with a valid authentication, while p' denotes a packet that has been modified by an attacker. The execution function $\psi_{\texttt{Upd}}$ (depicted below task `Upd`) defines that

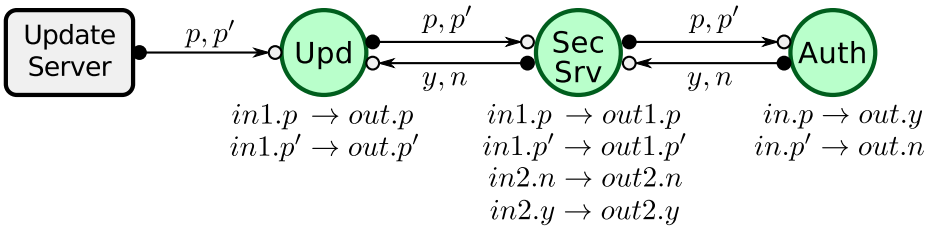

Fig. 2. Task Network for Update Service

such packets are simply forwarded to the security service task `SecSrv`. The same forwarding takes place at task `SecSrv`. The authentication function, i.e., `Auth`, of the security module is assumed to return whether incoming packets are valid or not. This behavior is modeled by the execution function $\psi_{\texttt{Auth}}$ that returns a y (yes) packet when the function has been called with a p packet, and n (no) otherwise. The answer is in advance relayed back to the update service function.

3.2 Towards Integration into Security Theory

Conventionally, a secure system is designed as follows: A set of security requirements is specified along with an attacker model with its capabilities before deploying the security protocols to achieve security. A rich set of theories exists for the verification of security protocols [4]. For example, based on a notion of protocols defining message transfers, term rewriting, and deduction rules; one can specify the abilities of an attacker and formally verify in advance whether the security protocol is vulnerable for the specified attacks [8,9].

Various well-established formalisms have been exploited for security analysis (such as CSP [11] and Petri nets [5]). As none of them was originally developed for architectural design, they have little relevance in actual system design for this design phase. Hence, our aim is to exploit real-time task networks, which is a well-established formalism that considers applications when they are deployed to a hardware architecture, for security analysis.

In order to model system functionality correctly, this clearly must involve the applied security protocols, as they might have a significant impact on the functionality and timing of the system. The proposed real-time model and its extensions [3] can be embedded in a natural way into security theories. This indeed could be done by relating our formalism to a respective CSP program [7] and then to apply security analysis as for example shown in [11]. For the present work however, we directly relate our formalism to the well-established Strand Space formalism developed in [13].

The Strand Space model exploits a basic notion of messages and message sequences:

Definition 2 ([13]). *A* signed term *is a pair* $\langle d, a\rangle$ *with* $a \in \mathbf{A}$*, and* $d \in \{+, -\}$*. We will write a signed term as* $+a$ *or* $-a$*.* $(\pm\mathbf{A})^*$ *is the set of*

finite sequences of signed terms. We will denote a typical element of $(\pm\mathbf{A})^*$ *by* $\langle\langle d_1, a_1\rangle, \ldots, \langle d_n, a_n\rangle\rangle$. ◇

Furthermore, the formalism defines a notion of protocols that are based on *strands*, which are essentially elements of $(\pm\mathbf{A})^*$:

Definition 3 ([13]). *A* Strand Space *is a set* S *with a trace mapping* $tr : S \to (\pm\mathbf{A})^*$. ◇

Given a task network N, it is easy to see how this relates to the above definitions. Considering the elements in the set Σ as messages that are transmitted between the protocol agents, provides us with the ground set of a Strand Space model: $\mathbf{A} = \Sigma$. Whether a message is received (denoted by $-$) or sent (denoted by $+$) can be obtained from the fact whether the message is observed at an input port or an output port respectively:

$$d(a) = \begin{cases} - & \text{if} \quad \exists t = (P_i, \psi, P_o) \in T : E(a) \in P_i \\ + & \text{if} \quad \exists t = (P_i, \psi, P_o) \in T : E(a) \in P_o \end{cases}$$

The translation gives rise to the integration of various important aspects of a system design. While the task network model is exploited for functional and real-time analysis, the translation allows the combination with security analysis. Based on the set $(\pm\mathbf{A})^*$ obtained by the translation, the security engineer is able to define the respective strands and bundles which form the basis (together with an understanding of the attacker capabilities) for the subsequent analysis. The main advantage of this integration is that all participants of the design have a common understanding of the design semantics.

The relation between the task network model and security formalisms as shown above is however, only a small part of what is needed in order to perform security analysis. In the following we concentrate on the notion of protocols, which is a crucial ingredient for security protocol analysis.

As said before, the Strand Space formalism defines the protocol sessions under consideration by finite sequences over messages, i.e., the elements of $tr(S)$. The causal and dependency relations, denoted by"$\rightarrow$" and "$\Rightarrow$" respectively, are essential for the formalism. They are also crucial for defining semantics of sending and receiving messages in a task network. Suppose a trace $\langle\langle d_1, m_1\rangle, \ldots, \langle d_n, m_n\rangle\rangle \in (\pm\mathbf{A})^*$. The reception of message m_i at the input port of a task, which corresponds to $\langle -, m_i\rangle$ in the trace, and the corresponding message sent by another task (which relates to $\langle +, m_i\rangle$) are causal related, i.e. we have the relation $\langle +, m_i\rangle \rightarrow \langle -, m_i\rangle$ in the corresponding Strand Space model. Additionally, the execution function of a task gives rise to the "$\Rightarrow$" relation of the corresponding Strand Space model. Given an actor (task) $t = (P_i, \psi, P_o)$, the dependency ($\Rightarrow$) relation of the strand is defined as:

$$\langle -, m_i\rangle \Rightarrow \langle +, m_{i+1}\rangle \quad \Longleftrightarrow \quad E(m_i) \in P_i \wedge E(m_{i+1}) \in P_o \wedge m_{i+1} = \psi(m_i)$$

4 Protocols and Verification

The relation proposed above is indeed only an initial step towards a more deeper integration of formal and security analyses. Nothing has been said for example about the capabilities of the intruder. More important, whether this would be possible without modifications of the underlying model in any case remains an open question. Explicit modeling of an intruder is not a well-established design step in current design processes for embedded systems.

It also remains open how the strand spaces for the security analysis of a given system are obtained. They essentially define the domain for the particular analysis. In the following we sketch the general approach for obtaining strand spaces from functional models.

Embedded systems are typically static systems. Dynamic behavior such as the addition of new functionality at run-time is rather uncommon. This is also reflected when using security protocols. We expect to see in a typical embedded system design only a fixed set of possibly available protocol behaviors. This fact can be exploited for deriving the parts of a security protocol actually used in a design in order to confine security analysis, and it should be possible to derive the relevant protocol behavior.

The security module in Figure 1, for example, provides a large set of possible protocol behaviors for authentication. In the depicted scenario however, only a single protocol instance is used, which is based on authenticating each incoming packet separately. This results in two possible protocol instances for the security protocol employed in the update service as shown in Figure 2, consisting of either the sequence `SecSrv` - p - `Auth` - y - `SecSrv` or `SecSrv` - p' - `Auth` - n - `SecSrv` between the `SecSrv` task (i.e., security service function) and the `Auth` block (i.e., authentication function).

The process of deriving the protocol behavior that is relevant for security analysis can also be done by exploiting formal analysis. For example, work in [3] proposes a translation scheme for our model into the formalism of timed automata [1]. Based on available model-checking tools this paves the way to obtain, for example, the relevant Strands for a given design model.

5 Conclusion

The goal of this contribution is to provide the basic notions for an integrated evaluation of the functional, real-time, and security aspect in the embedded system design utilizing formal mechanisms. For this, a functional model representing an example real-time system is considered as the target system and is modeled in the task network formalism. Then a possible relation is drawn between this formalism and the existing security theory, i.e., Strand Space formalism. However, drawing such a relation is only an initial step towards a more tighter integration between them. At the end, the presented formal approach is analyzed with an example to derive a protocol like behavior which is relevant for security analysis.

Acknowledgement. This work was supported by the Federal Ministry for Education and Research (BMBF) Germany, under grant code 01IS110355M, in project "Automotive, Railway and Avionic Multicore System (ARAMiS)" The responsibility for the content of this publication lies with the author.

References

1. Alur, R., Dill, D.L.: A Theory of Timed Automata. Theoretical Computer Science 126(2), 183–235 (1994)
2. Benveniste, A., Berry, G.: The Synchronous Approach to Reactive and Real-Time Systems. Proceedings of the IEEE 79, 1270–1282 (1991)
3. Büker, M., Metzner, A., Stierand, I.: Testing real-time task networks with functional extensions using model-checking. In: Proc. 14th IEEE International Conference on Emerging Technologies & Factory Automation, pp. 564–573. IEEE Press (2009)
4. Cortier, V., Kremer, S. (eds.): Formal Models and Techniques for Analyzing Security Protocols. IOS Press (2011)
5. Crazzolara, F., Winskel, G.: Petri nets in cryptographic protocols. In: Proceedings of the 15th International Parallel & Distributed Processing Symposium, IPDPS 2001, p. 149. IEEE Computer Society, Washington, DC (2001), http://dl.acm.org/citation.cfm?id=645609.662336
6. Denning, D.E.: A lattice model of secure information flow. Communications of the ACM 19(5), 236–243 (1976)
7. Faber, J., Stierand, I.: From high-level verification to real-time scheduling: A property-preserving integration. Reports of SFB/TR 14 AVACS 19, SFB/TR 14 AVACS (May 2007), iSSN: 1860-9821, http://www.avacs.org
8. Fröschle, S.: Adding branching to the strand space model. Electron. Notes Theor. Comput. Sci. 242(1), 139–159 (2009)
9. Fröschle, S., Sommer, N.: Reasoning with Past to Prove PKCS#11 Keys Secure. In: Degano, P., Etalle, S., Guttman, J. (eds.) FAST 2010. LNCS, vol. 6561, pp. 96–110. Springer, Heidelberg (2011)
10. Ghamarian, A.H., Geilen, M.C.W., Basten, T., Theelen, B.D., Mousavi, M.R., Stuijk, S.: Liveness and boundedness of synchronous data flow graphs. In: FMCAD 2006: Proceedings of the Formal Methods in Computer Aided Design, pp. 68–75. IEEE Computer Society, Washington, DC (2006)
11. Lowe, G.: Analysing Security Protocols Using CSP. In: Cortier, Kremer (eds.) [4] (2011)
12. Malipatlolla, S., Stierand, I.: Evaluating the Impact of Integrating a Security Module on the Real-Time Properties of a System. In: Schirner, G., Götz, M., Rettberg, A., Zanella, M.C., Rammig, F.J. (eds.) IESS 2013. IFIP AICT, vol. 403, pp. 343–352. Springer, Heidelberg (2013)
13. Fabrega, F.J.T., Herzog, J.C., Guttman, J.D.: Strand spaces: why is a security protocol correct? In: IEEE Symposium on Security and Privacy, pp. 160–171 (1998)
14. Tindell, K.W., Burns, A., Wellings, A.J.: Allocating hard real-time tasks: An NP-Hard problem made easy. Real-Time Systems 4, 145–165 (1992)
15. Yip, A., Wang, X., Zeldovich, N., Kaashoek, M.F.: Improving application security with data flow assertions. In: Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles, pp. 291–304. ACM (2009)