Employing the Concept of Multilevel Security to Generate Access Protection Configurations for Automotive On-Board Networks

Tobias Dörr*, Timo Sandmann*, Hannes Mohr†, and Jürgen Becker*
*Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany
Email: {tobias.doerr, sandmann, becker}@kit.edu
†ERNW Enno Rey Netzwerke GmbH, Heidelberg, Germany
Email: hmohr@ernw.de

***Abstract*—Future automotive on-board networks are expected to integrate various functions on only a few centralized processing platforms. Combined with attack surfaces that originate from the external connectivity of modern vehicles, this turns the design of secure on-board networks into a challenging endeavor. Therefore, we present a formal model to describe both confidentiality and integrity requirements of applications in such a network using security levels. The proposed model is then integrated into an existing design methodology for the automatic configuration of access protection units in MPSoCs. The resulting methodology ensures that the above-mentioned requirements are automatically enforced during runtime and is validated using a safety-critical example scenario from the automotive domain.**

***Index Terms*—Multilevel security, in-vehicle networks, multiprocessor system-on-chip, confidentiality, integrity.**

I. Introduction

Modern vehicles are often equipped with up to a hundred electronic control units (ECUs) interconnected via an on-board network [1]. From a cybersecurity perspective, the steadily increasing external connectivity of these systems leads to significantly wider attack surfaces [2]. In 2015, for instance, researchers were able to remotely access the main Controller Area Network (CAN) of a production vehicle by exploiting a vulnerability in its head unit [3]. In the automotive context, such vulnerabilities can be exploited to cause physical harm to humans or the environment, financial damage, operational disruptions, and privacy breaches [4].

A strict physical partitioning of the on-board network can help protect vehicles against successful attacks, but it is possible only to a certain extent as the ECUs are generally required to exchange data [1]. In fact, the ongoing consolidation of computing resources will even increase the need for on-board connectivity [5]: Future electric/electronic (E/E) architectures are expected to be centralized in the sense that a limited number of powerful processing platforms, often based on a multiprocessor system-on-chip (MPSoC), will tightly integrate a variety of different functions [6]. In the design of such systems, on-chip interconnects should therefore be treated as an integral part of the on-board network.

A design methodology that is able to facilitate such an approach was presented in [7]. It is targeted at networks of MPSoCs in which communication links—primarily the on-chip interconnects—are equipped with hardware entities to enforce a certain access policy during runtime. Such an entity is referred to as *access protection unit* (APU) and can be configured with a specification of permitted transactions. An example of a commercially available APU is the Xilinx Peripheral Protection Unit (XPPU) on the Zynq UltraScale+ MPSoC from Xilinx [8]. In the above-mentioned methodology, designers make use of a formal model to describe most importantly

1) the on-chip and off-chip architecture of the network,
2) applications running on the available execution units,
3) accepted information flows between applications, and
4) messages that the applications exchange.

From a model instance, the toolchain automatically derives APU configurations that are as prohibitive as possible but allow all specified messages that need to traverse a communication link to be transmitted. Under the assumption that all generated configurations are applied to their respective APU, the toolchain performs a static analysis to determine all potentially feasible information flows and compares them to the information flows that are specified as acceptable.

We consider such a fine-grained control of information flows a powerful mechanism to achieve confidentiality and integrity. However, the existing model is based on a rigorous whitelist approach in which every permitted information flow between a pair of applications needs to be explicitly specified. For highly complex on-board networks, the need for such an explicit specification can be a serious restriction.

We propose a design methodology that is based on the concept of multilevel security and integrate it into the design methodology from [7] in such a way that an explicit list of accepted information flows does no longer need to be provided as an input. Instead, the list is implicitly derived from security level annotations applied to certain design entities.

This work is organized as follows: Section II gives an overview of the proposed concept, while Section III summarizes the original design methodology from [7]. We present a suitable model extension in Section IV, describe its implementation and validation in Section V, and close this paper with a discussion of related work in Section VI.

II. Background and Motivation

This section gives a short summary of relevant multilevel security approaches and motivates their application to the design of secure automotive on-board networks.

A. Preliminaries on Multilevel Security

The concept of *multilevel security* (MLS) in computer systems dates back to the early 1970s, when government organizations faced the challenge that machines were required to handle pieces of information with varying security classifications. To tackle this issue, Bell and LaPadula proposed a model of a security policy to preserve confidentiality in such systems [9]. It is based on *subjects*, which represent processes, and *objects* describing passive entities such as data, files, and I/O devices. Subjects and objects are assigned a *security level* that consists of a *classification* from a totally ordered set (such as Unclassified or Top Secret) and an arbitrary number of *categories* (such as Nuclear or Crypto).

In this model, a security level a is said to *dominate* security level b if and only if the classification of a is at least as high as the classification of b and the category set of a includes the category set of b. A subject is only allowed

- to observe an object if its security level dominates the security level of the object ("no read up") and
- to modify an object if its security level is dominated by the security level of the object ("no write down").

This model serves, for instance, as the basis of the MLS mode implemented by SELinux [10]. Note, however, that it is not concerned with and therefore unable to preserve the integrity of a shared computer system.

Biba [11] proposed a model that is based on the concept of *integrity levels*. In a particular manifestation, it requires the enforcement of the "no write up" and "no read down" property with respect to these integrity levels. This policy is able to preserve the integrity of a shared computer system, but it is not concerned with confidentiality. It can therefore be seen as a complement to the Bell-LaPadula model.

In the following, we will refer to security levels from the Bell-LaPadula model as *confidentiality levels* and use the term *security level* to refer to both kinds of levels.

B. Overview of the Proposed Concept

In this work, we present a methodology for the model-based design of on-board networks that applies adaptations of the Bell-LaPadula and the Biba model to ensure that resulting systems meet their confidentiality and integrity requirements by construction. An adaptation of the approaches is necessary because such networks are inherently distributed and therefore fundamentally different than a single computer system. While a traditional operating system is in control of all its subjects and objects, information in an on-board network might traverse multiple independent on-chip and off-chip nodes before reaching its destination. Therefore, we reformulate the access policies from the models in a manner that captures a possible propagation through the network and is independent of individual transactions. The policies resulting from this can roughly be described as follows:

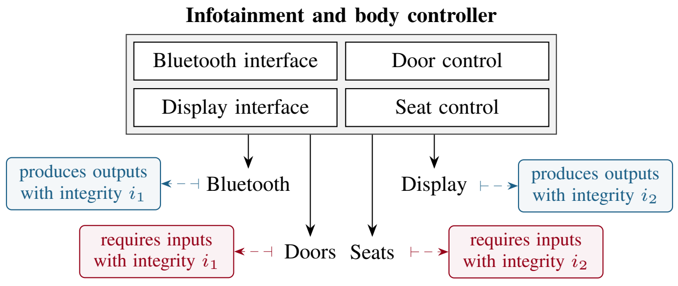

Infotainment and body controller
Bluetooth interface
Door control
Display interface
Seat control
produces outputs with integrity i_1
Bluetooth
Display
produces outputs with integrity i_2
requires inputs with integrity i_1
Doors
Seats
requires inputs with integrity i_2

Fig. 1. Controller with its attached components

- Confidentiality is preserved ⇔ Information from a source that requires a certain confidentiality level must not flow to a sink that provides a lower confidentiality level.
- Integrity is preserved ⇔ Information from a source that provides a certain integrity level must not flow to a sink that requires a higher integrity level.

A more precise definition of these properties will be given along with the description of the concept in Section IV.

Example 1. *Consider an artificial E/E architecture that comprises the combined infotainment and body controller shown in Fig. 1. This controller is assumed to be implemented on a single heterogeneous MPSoC and directly attached to the vehicle's primary touch display ("Display") as well as to a wireless module ("Bluetooth") that allows users to connect to the vehicle using a suitable device such as a smartphone. The controller is further attached to two independent off-chip buses (such as two CAN buses) that connect it to the doors and the seats of the vehicle, respectively. Assume that the following functional and non-functional requirements have been specified for this subsystem:*

1) *Passengers shall be able to adjust the seats and control the door locks from the touch display.*
2) *Authorized users shall be able to control the door locks from outside the vehicle via Bluetooth.*
3) *For safety reasons, the driver seat must be adjustable only from within the vehicle, i.e., not via Bluetooth.*

To meet the first two requirements, the controller executes the four applications shown in the figure as operating system tasks, for instance. They interact with each other and their respective peripheral in a suitable manner. For instance, the "Bluetooth interface" application listens for user requests from the Bluetooth module, forwards them to the "Door control" application, which in turn interacts with the individual doors. The third requirement can be modeled by defining the integrity levels i_1 and i_2 as follows:

- *i_1 represents all possible kinds of user input, while*
- *i_2 represents user input from within the vehicle.*

Here, i_2 is "at least as trustworthy" as i_1 and therefore dominates i_1. The concept proposed in this paper allows the *designer to label certain design entities (such as the Bluetooth module or the doors) as providing or requiring a certain integrity level. For the purposes of this example, the following labeling is performed and can also be seen from Fig. 1:*

- *Since the Bluetooth module introduces remote input into the on-board network, it is labeled as producing i_1 output.*
- *Since the display outputs originate from within the vehicle, they are labeled as having an integrity of i_2.*
- *The seats must not be accessible from outside the vehicle and are therefore labeled as requiring inputs with an integrity level of at least i_2.*
- *The doors are labeled as requiring inputs with an integrity level of at least i_1.*

Most importantly, this implies that an information flow from the Bluetooth module to an entity that requires i_2 inputs is not possible. If the implementation of the on-board network is performed in such a manner that this "contract" is satisfied, a remote attacker that obtains access via the Bluetooth module remains unable to interfere with the seats. It is crucial to understand that the goal of the presented approach is not to prevent exposed applications such as the Bluetooth interface from becoming compromised. Instead, it supports designers to construct on-board networks in which the impact of feasible attacks is reduced to a tolerable level.

Similar considerations can be made for the confidentiality case. Nevertheless, it is important to understand that integrity and confidentiality levels can be defined and annotated to model entities independently of each other. Furthermore, note that this process can be performed without having to consider low-level hardware details or possible attack paths.

III. Description of the Original Model

The design methodology presented in [7], which forms the foundation of this work, is based on a model that captures the relevant aspects of a system under consideration in a formal manner. We refer to it as the *original model* and will give a short description of its structure in this section.

The original model consists of three layers and is summarized in Table I, where $\mathcal{P}(\cdot)$ denotes the power set. The table presents details not relevant for the purposes of this paper in a simplified manner. Most importantly, it merges similar definitions from various layers of the original model and, therefore, lists them as not belonging to a specific one.

The lowest layer (I) captures the *platform architecture*. It consists of a set of units U, a set of links L, and a function φ_L mapping every link to the set of units that it is attached to. In general, an example of a unit is an I/O controller or an on-chip processing core along with its real-time operating system. Using the δ function, units can be marked as *dependable*. Dependable units must be designed in such a manner that they fulfill certain requirements. Most importantly, they are expected to isolate applications from each other in such a way that no unintended information flow between them is possible, even if the underlying hardware is affected by random faults. A fault-tolerant processing core executing a

TABLE I
Relevant Definitions from the Original Model

Layer	Model element	Description
I	$U = \{u_1, u_2, \ldots\}$	Execution units (*units*) including their runtime environments.
	$L = \{\ell_1, \ell_2, \ldots\}$	On-chip and off-chip communication links (*links*) between units.
	$\varphi_L \colon L \to \mathcal{P}(U)$	Mapping of all links to the units that are attached to them.
II	$F_T = \{t_1, t_2, \ldots\}$	*Terminal features* as sources and sinks of information flow.
	$I \subseteq F_T \times F_T$	Specification of all accepted information flows.
III	$F_F = \{f_1, f_2, \ldots\}$	*Forwarding features* to pass on information to other features.
	$X_R, X_W \subseteq F \times L \times F$	Envisaged read (R) and write (W) transactions over links.
	$X_L \colon F \times F$	Information flows implemented locally, i.e., within units.
	$\omega_L \colon L \to \{0,1\}$	Whether or not to protect a specific link using an APU.
N/A	$F = F_T \cup F_F$	Union of all terminal and forwarding features.
	$\delta\colon (U \cup F) \to \{0,1\}$	Mapping of all units and features to their *dependability*.
	$\sigma \colon F \to U$	Mapping of each feature to the unit that executes it.

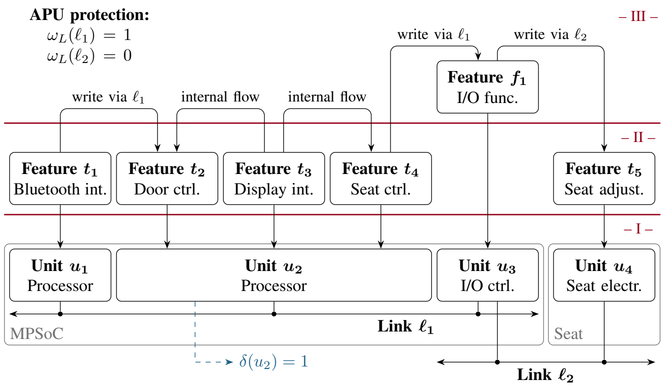


Fig. 2. Sample instance of the three-layered original model

trusted operating system or hypervisor is an example of a unit that can be marked as dependable. Dependability declarations are automatically taken into account during the determination of all potentially feasible information flows.

Example 2. *A sample instance of such a layer-I description is visualized in the lowermost portion of Fig. 2, in which the red horizontal lines separate the three model layers from each other. This platform architecture is based on the scenario described in Example 1 but, for demonstration purposes, artificially limited to four units and two links. More specifically, it consists of the units $U = \{u_1, u_2, u_3, u_4\}$ and the communication links $L = \{\ell_1, \ell_2\}$, where ℓ_1 is an* *on-chip interconnect attached to u_1, u_2, and u_3, while ℓ_2 is an off-chip interconnect (such as a CAN bus) attached to u_3 and u_4. For the on-chip I/O controller u_3, for instance, this attachment is formally expressed as $\varphi_L(u_3) = \{\ell_1, \ell_2\}$. The on-chip processing core u_2 is declared to be dependable, while all other units are assumed to be undependable. We further assume that u_1 has direct and private access to a dedicated Bluetooth module, while u_2 has direct access to the touch display and all doors of the vehicle. However, these components as well as additionally relevant I/O controllers are not modeled as part of this example to limit its complexity.*

The next layer (II) captures the *functional implementation*. Here, every application (such as a task executed by an operating system) is described as a *terminal feature*. The set of all terminal features is referred to as F_T. Every terminal feature is mapped to the unit that delivers it via the σ function. Just as units, terminal features can be marked as dependable using the δ function. Dependable terminal features are expected to be designed in such a manner that information they receive is only passed on in a strictly controlled manner. An example of a dependable terminal feature is one that does not contain systematic faults, applies suitable fault tolerance mechanisms, and cryptographically verifies all its inputs. Furthermore, this layer comprises the relation I, which describes all accepted information flows between terminal features. It represents the previously described whitelist that all potentially feasible information flows are compared against.

Example 3. *In continuation of Example 2, we define the set of terminal features as $F_T = \{t_1, t_2, t_3, t_4, t_5\}$ and use the σ function to map them to units as visualized in Fig. 2. The set of all accepted information flows is defined as*

$$I = \{(t_1, t_2), (t_3, t_2), (t_3, t_4), (t_3, t_5), (t_4, t_5)\}.$$

This corresponds to the most prohibitive assignment that allows all functional requirements to be implemented. Since we expect t_3 to be able to influence t_4, t_4 to be able to influence t_5, and due to the fact that t_4 is undependable, we must also accept a flow from t_3 to t_5. In practice, the designer is responsible to ensure that I contains only elements that are in line with the desired information flow policy.

At the highest layer of the model (III), designers specify the *information flow implementation* to describe how exactly individual terminal features exchange information. *Forwarding features* defined in the table are comparable to terminal features but have the sole purpose of forwarding or buffering every piece of information they receive. The functionality of a shared memory or an I/O controller, for instance, can be modeled as a forwarding feature. We refer to the set F, which contains the union of all terminal and forwarding features, as the set of *features*. X_R and X_W capture the actual read and write transactions that the designer intends to implement using the modeled communication links. Similarly, X_L is used to describe all flows that shall be realized internally within a unit, i.e., without involving a communication link. Finally, the

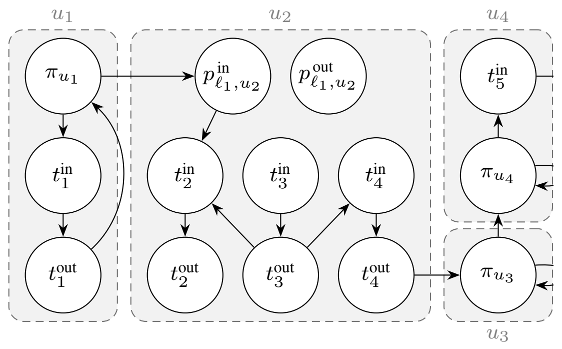


Fig. 3. Excerpt of G_β for the sample model instance

designer uses the ω_L function to decide on a per-link basis whether or not an APU configuration shall be generated for a certain communication link $\ell \in L$.

In the design methodology, a model instance is used to conduct an *information flow analysis*. This step assumes that during runtime, all links $\ell \in L$ for which an APU protection is requested ($\omega_L(\ell) = 1$) enforce access protection rules that are as prohibitive as possible but allow all transactions from X_R and X_W to occur. Based on this assumption and the model instance, a *beta graph* (G_β) is constructed. Most importantly, it contains an input and an output node (t^{in} and t^{out}) for every terminal feature $t \in F_T$. Edges and helper nodes (such as unit sharing nodes π_u, unit-link input ports $p^{\text{in}}_{\ell,u}$, and unit-link output ports $p^{\text{out}}_{\ell,u}$) are added in such a way that the resulting graph reflects all potentially feasible information flows. More specifically, a directed path from a t^{out} to a t^{in} node of G_β represents a potential end-to-end information flow between the respective terminal features. The toolchain verifies that all such flows are in line with I and, if this is the case, generates platform-specific APU configurations that can be deployed to the hardware to enforce exactly the access policy that the static analysis assumed to be enforced. In case of a discrepancy between I and the paths in G_β, the designer has to refine the model instance until the discrepancy is eliminated.

Example 4. *We continue Example 3 by extending the partially complete model instance with the definitions shown in the uppermost portion of Fig. 2. More specifically, we introduce the forwarding feature $f_1 \in F_F$ to represent the functionality of the I/O controller connecting the MPSoC to the off-chip bus. We further request an APU configuration to be generated for the on-chip interconnect ℓ_1 and populate X_W, W_R, and X_L with values representing the interactions shown in the figure. For instance, X_W contains the tuple (t_4, ℓ_1, f_1) to capture that the implementation of the "Seat control" task uses the on-chip interconnect of the MPSoC to issue write transactions targeted at f_1. An excerpt of the G_β that the information flow analysis procedure generates for the complete model instance is shown in Fig. 3. All directed paths from a t^{out} to a t^{in} node, such as from t_4 to t_5, are part of I, the relation that describes all accepted information flows. Therefore, the toolchain will automatically generate an APU configuration for ℓ_1. In this* *specific case, the configuration will for instance prevent u_1 from directly writing to u_3 via the on-chip interconnect. Such a transaction would especially allow for an information flow from t_1 to t_5, which is considered unacceptable.*

IV. Definition of the Model Extension

In this section, we replace I with new model elements that allow a designer to capture the desired information flow policy without having to specify an explicit and exhaustive list of all accepted end-to-end flows between terminal features. The presented extension consists of a confidentiality and an integrity framework. Although the frameworks share the same underlying idea, they are independent of each other and can also be used in isolation.

A. Confidentiality and Integrity Levels

The following definition is inspired by the Bell-LaPadula model [9] and the terminology used in SELinux [10].

Definition 1. *A* security level *is the combination of a* sensitivity *from the totally ordered set*

$$S = \{s_1, s_2, \dots, s_n\} \quad \textit{with} \quad s_n \geq s_{n-1} \geq \dots \geq s_1$$

and an arbitrary number of categories *from the set*

$$K = \{k_1, k_2, \dots, k_m\}.$$

Here, $s_i \geq s_j$ means that s_i is "at least as sensitive" as s_j. The set of all possible security levels is then given by

$$A = S \times \mathcal{P}(K),$$

where $\mathcal{P}(K) = \{U \mid U \subseteq K\}$ denotes the power set of K. We adopt the definition from the Bell-LaPadula model and say that the level $a_i = (s_i, K_i) \in A$ dominates $a_j = (s_j, K_j) \in A$ if and only if $s_i \geq s_j$ and $K_i \supseteq K_j$. In the following, this relationship will also be expressed as $a_i \propto a_j$.

Mathematically, $(A, \propto)$ forms a lattice in the sense that there are binary operations to return the join ($\vee$) and meet ($\wedge$) of any two given security levels. In the specific case of $(A, \propto)$, the join is obtained by choosing the maximum of the sensitivities and unifying the category sets of the two security levels that are combined. Analogously, the meet is calculated by choosing the minimum of the sensitivities and intersecting the category sets of the combined security levels. Therefore, these operations are commutative and associative. Furthermore, there is a maximum security level, which dominates all others, and a minimum security level, which is dominated by all others: $\max(A) = (s_n, K)$ and $\min(A) = (s_1, \varnothing)$.

Our methodology allows a dedicated set of security levels to be defined for each framework. Depending on the framework that it originates from, we refer to a security level as either a *confidentiality level* or an *integrity level*. To avoid ambiguities, we add a superscript "c" to confidentiality-related symbols and a superscript "i" to integrity-related symbols.

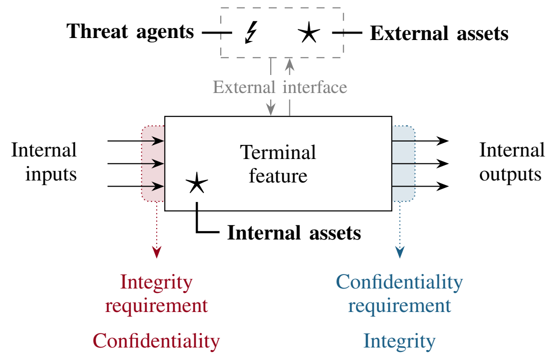


Fig. 4. Assets, inputs, and outputs of a terminal feature

Example 5. *Consider a specific confidentiality framework that the designer of an on-board network has defined as follows:*

$$S^c = \{s_1^c, s_2^c, s_3^c\} \quad \textit{with} \quad s_3^c \geq s_2^c \geq s_1^c,$$
$$K^c = \{k_A^c, k_B^c, k_C^c, k_D^c\}.$$

The set of all possible confidentiality levels $A^c = S^c \times \mathcal{P}(K^c)$ will then contain $|A^c| = 3 \cdot 2^4 = 48$ elements, including the levels $a_1^c = (s_2^c, \varnothing)$, $a_2^c = (s_3^c, \{k_B^c\})$, and $a_3^c = (s_2^c, \{k_A^c, k_C^c\})$. Note that that the latter two levels dominate the first one, but there is no dominance relationship between a_2^c and a_3^c. Furthermore, observe that the following equations hold:

$$a_1^c \vee a_2^c = a_2^c, \quad a_1^c \vee a_3^c = a_3^c, \quad \textit{and}$$
$$a_2^c \vee a_3^c = (s_3^c, \{k_A^c, k_B^c, k_C^c\}).$$

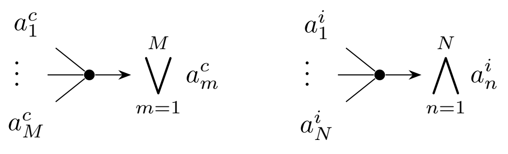
B. Annotation and Propagation of Security Levels

The original design methodology from [7] is based on the idea that features receive inputs from other features and deliver outputs that are in turn consumed by other features from the model instance. It does not allow designers to distinguish individual inputs and outputs of a specific feature. Instead, every feature has exactly one pool of inputs and one pool of outputs.[1] They are represented by the t^{in} and t^{out} nodes of G_β. For the purposes of this paper, we refer to these pools as the *internal inputs* and the *internal outputs* of a feature, respectively. Furthermore, the original design methodology assumes that every terminal feature has certain properties or capabilities (such as being able to perceive the environment or to influence the environment) that the designer considers during the derivation of the I relation. In this work, we refine and formalize this aspect as follows: In addition to its internal inputs and outputs, every terminal feature is associated with an arbitrary number of *internal assets*. Furthermore, it is associated with an arbitrary number of *external assets* that it has access to via its external interface (see Fig. 4). Internal assets are integrated into the terminal feature itself and are valuable from a confidentiality or an integrity perspective. External assets are accessible entities from the environment of

[1]If necessary, an application or task needs to be decomposed into as many features as are necessary to achieve the desired granularity.

TABLE II
IMPACT OF ASSETS AND THREAT AGENTS OF A TERMINAL FEATURE

Entities	Impact on the model instance	
	Confidentiality	Integrity
Internal and external assets	Determine the confidentiality that is **required** at the internal **outputs**.	Determine the integrity that is **required** at the internal **inputs**.
Threat agents	Determine the confidentiality that is **provided** at the internal **inputs**.	Determine the integrity that is **provided** at the internal **outputs**.



(a) Confidentiality (b) Integrity

Fig. 5. Resulting security levels for converging information flows

the system under consideration that are again valuable from a confidentiality or an integrity perspective.

Example 6. *Consider the seat adjustment functionality that is part of the model instance visualized in Fig. 2. It is represented by the terminal feature t_5 and, using an integrated actuator, able to alter the position of the driver's seat. Therefore, an attack to this feature has the potential to interfere with driver's ability to control the vehicle. Under certain conditions (such as at high speeds), this attack might lead to an accident and, consequently, cause physical harm to humans. From the perspective of t_5, this capability can be considered as an external asset that is relevant from an integrity point of view.*

Example 7. *The door control feature visualized in Fig. 2 might be implemented as an operating system task containing a private key that the car manufacturer uses to communicate with the individual doors in an secure manner. Since it must be kept secret, this key can be seen as an internal asset of the task that is relevant from a confidentiality point of view.*

However, terminal features are not only associated with the assets that need to be protected. Their external interfaces are also the paths that *threat agents* might make use of in order to attack the system's assets (see Fig. 4). More specifically, threat agents might try to use the terminal feature in order to gain unauthorized access to information from the on-board network or affect the on-board network in an unauthorized manner.

Based on all these considerations and the original design methodology, we propose the following four-step approach to prevent such attacks by design:

a) Security level definition: Before layer II of the model instance is populated, suitable sets of confidentiality and/or integrity levels are defined by the designer.

b) Security level annotation: The designer evaluates every envisaged terminal feature individually to derive required and provided security levels at the internal inputs and internal outputs of the feature (see again Fig. 4). While doing so, assets and threat agents of the feature under consideration must be considered and will generally impact the resulting security levels as described in Table II. For terminal features with $\delta(\cdot) = 0$, the manner in which the internal outputs are derived from the internal inputs is irrelevant during this step. This is due to the fact that the potential for an unintended information flow from its internal inputs to its internal outputs is already reflected in G_β. Dependable terminal features, on the other hand, are special "trust anchors" for which this assumption would be overly pessimistic. The manner in which they derive their internal outputs from their internal inputs therefore needs to be carefully assessed by the designer and incorporated into the annotation of security levels.

c) Security level propagation: As the first step of the information flow analysis, security levels from all internal outputs of terminal features are propagated along the edges of G_β. Whenever potentially feasible information flows converge during this propagation, they are handled as visualized Fig. 5.

d) Information flow verification: Finally, it is verified that the provided confidentiality at the internal inputs of every $t \in F_T$ is sufficiently "high" and that the required integrity at the internal inputs of every $t \in F_T$ is sufficiently "low", both with respect to the propagated security levels.

Before we describe the information flow analysis in more detail in Section V, we focus on the second step and formalize the security level annotation procedure as follows.

Definition 2. *The confidentiality framework is based on a set of confidentiality levels, A^c, and expects a* confidentiality requirement *as well as a* confidentiality *to be specified for every $t \in F_T$. Formally, this labeling is performed at layer II of the model by specifying the functions*

$$\lambda_R^c \colon F_T \to A^c \quad \textit{and} \quad \lambda_P^c \colon F_T \to A^c.$$

Here, λ_R^c describes the confidentiality requirement at all internal outputs, while λ_P^c refers to the provided confidentiality at all internal inputs. It is reasonable to set $\lambda_R^c(t) = \min(A^c)$ for all $t \in F_T$ that do not introduce confidential information into the system and $\lambda_P^c(t) = \max(A^c)$ for all $t \in F_T$ whose data is guaranteed to be unaccessible to threat agents.

Definition 3. *The integrity framework is based on a set of integrity levels, A^i, and expects an* integrity requirement *as well as an* integrity *to be specified for every $t \in F_T$. This labeling is performed by populating the layer-II functions*

$$\lambda_R^i \colon F_T \to A^i \quad \textit{and} \quad \lambda_P^i \colon F_T \to A^i.$$

λ_R^i captures the integrity requirement at all internal inputs, while λ_P^i describes the integrity levels provided at all internal outputs. In this case, $\min\left(A^i\right)$ and $\max\left(A^i\right)$ are neutral elements for λ_R^i and λ_P^i, respectively.

C. Remarks on Dependability and the Scope of this Work

In the original model, a feature $f \in F$ with $\delta(f) = 1$ can be mapped to an undependable unit. The information flow analysis handles this case by implicitly setting $\delta(f)$ to 0 to take unwanted effects of the untrustworthy execution platform into account. For our purposes, however, this implicit handling collides with the fact that dependable and undependable terminal features need to be handled differently during the security level annotation. In the extended model, dependable features can therefore only be mapped to dependable units.

Finally, it is important to understand that threat agents might try to attack an asset of the very terminal feature they use to access the system. Such an attack is limited to one terminal feature, unrelated to the on-board network per se, and therefore beyond the scope of this work.

V. Implementation and Validation

The new model does no longer contain the I relation. Instead, it comprises $(A, \propto)^c$, λ^c_R, and λ^c_P from the confidentiality framework as well as $(A, \propto)^i$, λ^i_R, and λ^i_P from the integrity framework. We incorporated the new model elements into the original design methodology as follows:

1) Using the capabilities of the Xtext framework, we developed a domain-specific language (DSL) that designers can use to describe instances of the extended model.
2) We integrated the G_β construction algorithm described in [7] into our toolchain. Since this algorithm does not depend on I, this was possible without modifications.
3) We implemented a refined information flow analysis algorithm that operates on the beta graphs to propagate the security levels as outlined in Section IV. This algorithm will be described in the next subsection in detail.
4) We implemented an adapter that automatically triggers the APU configuration generators from the original design methodology if and only if the information flow analysis procedure is successful.

A. Algorithm for the Information Flow Analysis

The information flow analysis procedure is shown in Algorithm 1 and will be executed twice by the developed toolchain: once with mode $m = c$ for the confidentiality case and once with mode $m = i$ for the integrity case. The information flow analysis procedure is successful if and only if both invocations return a "true" value (line 23).

The algorithm operates on two data structures: "levels" is a map that stores the security level propagated to every G_β node at any point in time, while the "dirtySet" keeps track of all the nodes for which a recently modified "levels" value has not been propagated to the successors of the respective node yet. Initially, the "levels" value of all nodes is set to the neutral element with respect to the mode (line 4). Confidentiality requirement and integrity annotations (λ^c_R and λ^i_P) from the model instance are then used to overwrite the respective "levels" entries (line 6). Furthermore, the "dirtySet" is initialized with all t^{out} nodes (line 7). Then, as long as the "dirtySet" is not empty, nodes from this set are visited. This means that a new security level is calculated for all its immediate successors as per Fig. 5. Furthermore, all modified successors of a visited node are added to the "dirtySet". At the end of this phase, the propagated confidentiality requirement of all nodes

Algorithm 1 Information flow analysis for $m \in \{c, i\}$

```
1: levels ← newMap($V \to A^m$)                              ▷ Initialization
2: dirtySet ← newSet($V$)
3: for all $v \in V$ do
4:     levels($v$) ← if $m = c$ then $\min(A^c)$ else $\max(A^i)$
5: for all $t^{\text{out}} \in V$ with $t \in F_T$ do
6:     levels($t^{\text{out}}$) ← if $m = c$ then $\lambda^c_R(t)$ else $\lambda^i_P(t)$
7:     dirtySet.add($t^{\text{out}}$)
8: while ¬dirtySet.empty() do                                 ▷ Propagation
9:     $x$ ← dirtySet.pop()
10:    for all $y$ in $G_\beta$.successors($x$) do
11:        if $m = c$ then
12:            newLevel ← levels($x$) $\vee$ levels($y$)
13:        else if $m = i$ then
14:            newLevel ← levels($x$) $\wedge$ levels($y$)
15:        if newLevel $\neq$ levels($y$) then
16:            levels($y$) ← newLevel
17:            dirtySet.add($y$)
18: for all $t^{\text{in}} \in V$ with $t \in F_T$ do          ▷ Verification
19:    if $m = c$ and $\neg(\lambda^c_P(t) \propto$ levels($t^{\text{in}}$)$)$ then
20:        return false
21:    if $m = i$ and $\neg($levels($t^{\text{in}}$) $\propto \lambda^i_R(t))$ then
22:        return false
23: return true
```

dominates that of all direct and indirect predecessors, while the propagated integrity of all nodes is dominated by that of all direct and indirect predecessors. Finally, the confidentiality and integrity requirement annotations (λ^c_P and λ^i_R) from the model instance are compared against these propagated levels. In both modes, the verification is successful if and only if the provided security levels dominate the required ones. This is verified in line 19 and 21, respectively.

B. Application to the Running Example

To validate the proposed concept and its implementation, we created various model instances, performed a manual analysis of potential attack paths, and derived from these considerations whether we expect a positive or a negative outcome from the information flow analysis. We then applied our implementation to these instances and ensured that the expectations were met. The following two paragraphs outline some of our findings regarding the running example.

Example 8. *Consider again Fig. 1 along with the model instance from Fig. 2. To capture relevant integrity aspects, we specified an A^i with $S^i = \{i_1, i_2\}$, $i_2 \geq i_1$, and an empty category set. As suggested in Example 1, we set*

$$\lambda^i_P(t_1) = (i_1, \varnothing), \qquad \lambda^i_P(t_3) = (i_2, \varnothing),$$
$$\lambda^i_R(t_5) = (i_2, \varnothing), \qquad \lambda^i_R(t_2) = (i_1, \varnothing),$$

and all other λ^i_P or λ^i_R values to their neutral elements. Our manual analysis revealed that in this specific scenario, no feasible attack paths that lead from t_1 to t_5, the safety-critical *seat adjustment feature, exist. We therefore expected the information flow analysis to succeed. In fact, the final integrity that propagated to the input node of t_5, was $(i_2, \varnothing)$. Since this provided integrity dominates the integrity requirement of t_5, i.e., $\lambda^i_R(t_5)$, the algorithm returned a "true" value.*

Example 9. *Based on the model instance from Example 8, we derived several slightly modified instances for which the information flow analysis should return a "false" value. If one no longer requests an APU protection for ℓ_1 by setting $\omega_L(\ell_1)$ to 0, for instance, no mechanism prevents u_1 from directly writing to u_3. Other possibilities to create a vulnerability that allows t_1 to influence t_5 in an unacceptable manner are to set $\delta(u_2)$ to 0 or to extend X_L with (t_2, t_3). In all these cases, the integrity that propagates to t_5^{in} is $(i_1, \varnothing)$. Since this security level does not dominate $(i_2, \varnothing)$, the information flow analysis returned "false" in all cases.*

VI. Related Work

Various approaches to secure automotive on-board networks against different types of attacks have been proposed. Examples include probabilistic methods to guide security engineers during the design of such networks [12], authentication [13] and traffic monitoring [14] schemes, as well as novel gateway architectures with isolation capabilities [15]. However, such approaches typically focus on system-level interconnects and neglect potential on-chip effects of attacks.

From an on-chip perspective, hardware units to control transactions on the shared interconnect [16] have been presented as a potential security measure and are comparable to APUs available on modern MPSoCs. The methodology from [7] builds up on this concept and combines it with a static analysis of how information is able to propagate through the system, which again has similarities to information flow tracking schemes such as the one presented in [17].

Our multilevel security extension pursues a similar goal as the taint tracking approach from [18]. However, the authors of this work propose a scheme based on the instrumentation of binary executables, while our approach is performed offline and eventually utilizes the platform-specific code generation capabilities from [7]. Lattice-based security levels have been applied to gate-level information flow tracking [19]. Compared to our work, however, this approach considers information flows in hardware circuits at a considerably lower level of abstraction. It is able to capture potential side-channel attacks but has limited applicability to software-centric systems such as future automotive on-board networks.

VII. Conclusion

The goal of this work was to present a methodology to design automotive on-board networks that fulfill their confidentiality and integrity requirements. Based on an existing design methodology to enforce end-to-end information flow policies in MPSoC-based systems, we showed that it is possible to derive the exact policies to enforce implicitly from a security level assignment that we consider to be more manageable than an explicit specification of accepted information flows.

In the proposed concept, security levels are based on sensitivities and categories. It is conceivable to allow them to be drawn from arbitrarily defined lattices. Trading off the flexibility that this creates against the increased tooling complexity is a topic for future research.

Acknowledgment

This work was funded by the German Federal Ministry of Education and Research (BMBF) under grant number 16KIS0886 (DEFEnD). The responsibility for the content of this publication lies with the authors.

References

[1] M. H. Eiza and Q. Ni, "Driving with Sharks: Rethinking Connected Vehicles with Vehicle Cybersecurity," *IEEE Vehicular Technology Magazine*, vol. 12, no. 2, pp. 45–51, 2017.

[2] R. Ernst, "Automated Driving: The Cyber-Physical Perspective," *Computer*, vol. 51, no. 9, pp. 76–79, 2018.

[3] C. Miller, "Lessons Learned from Hacking a Car," *IEEE Design & Test*, vol. 36, no. 6, pp. 7–9, 2019.

[4] SAE International, "J3061: Cybersecurity Guidebook for Cyber-Physical Vehicle Systems," 2016.

[5] O. Burkacky, J. Deichmann, and J. P. Stein, "Automotive software and electronics 2030," McKinsey & Company, Tech. Rep., Jul. 2019.

[6] M. Hassan, "Heterogeneous MPSoCs for Mixed-Criticality Systems: Challenges and Opportunities," *IEEE Design & Test*, vol. 35, no. 4, pp. 47–55, 2018.

[7] T. Dörr, T. Sandmann, and J. Becker, "A Formal Model for the Automatic Configuration of Access Protection Units in MPSoC-Based Embedded Systems," in *23rd Euromicro Conference on Digital System Design (DSD)*, Kranj, Slovenia, 2020.

[8] S. McNeil, P. Schillinger, A. Kolarkar, E. Puillet, and U. Gertheinrich, "Isolation Methods in Zynq UltraScale+ MPSoCs," 2019, Xilinx Application Note, XAPP1320.

[9] D. E. Bell and L. J. LaPadula, "Secure Computer System: Unified Exposition and MULTICS Interpretation," MITRE Corporation, Technical Report, Mar. 1976.

[10] B. Hicks, S. Rueda, L. St.Clair, T. Jaeger, and P. McDaniel, "A logical specification and analysis for SELinux MLS policy," *ACM Transactions on Information and System Security*, vol. 13, no. 3, 2010.

[11] K. J. Biba, "Integrity Considerations for Secure Computer Systems," MITRE Corporation, Technical Report, Jun. 1975.

[12] P. Mundhenk, S. Steinhorst, M. Lukasiewycz, S. A. Fahmy, and S. Chakraborty, "Security Analysis of Automotive Architectures using Probabilistic Model Checking," in *Proceedings of the 52nd Annual Design Automation Conference*, San Francisco, CA, USA, 2015.

[13] C.-W. Lin and A. Sangiovanni-Vincentelli, "Cyber-Security for the Controller Area Network (CAN) Communication Protocol," in *2012 International Conference on Cyber Security*, 2012.

[14] P. Waszecki, P. Mundhenk, S. Steinhorst, M. Lukasiewycz, R. Karri, and S. Chakraborty, "Automotive Electrical and Electronic Architecture Security via Distributed In-Vehicle Traffic Monitoring," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 11, pp. 1790–1803, 2017.

[15] S. Shreejith, P. Mundhenk, A. Ettner, S. A. Fahmy, S. Steinhorst, M. Lukasiewycz, and S. Chakraborty, "VEGa: A High Performance Vehicular Ethernet Gateway on Hybrid FPGA," *IEEE Transactions on Computers*, vol. 66, no. 10, pp. 1790–1803, Oct. 2017.

[16] T. Nojiri, Y. Kondo, N. Irie, M. Ito, H. Sasaki, and H. Maejima, "Domain Partitioning Technology for Embedded Multicore Processors," *IEEE Micro*, vol. 29, no. 6, pp. 7–17, 2009.

[17] J. Oberg, W. Hu, A. Irturk, M. Tiwari, T. Sherwood, and R. Kastner, "Information flow isolation in I2C and USB," in *Proceedings of the 48th Design Automation Conference*, San Diego, CA, USA, 2011.

[18] H. Schweppe and Y. Roudier, "Security and Privacy for In-Vehicle Networks," in *1st International Workshop on Vehicular Communications, Sensing, and Computing (VCSC)*, Seoul, South Korea, 2012.

[19] W. Hu, J. Oberg, J. Barrientos, D. Mu, and R. Kastner, "Expanding Gate Level Information Flow Tracking for Multilevel Security," *IEEE Embedded Systems Letters*, vol. 5, no. 2, pp. 25–28, 2013.