Formalising the Impact of Security Attacks on IoT Safety

Ehsan Poorhadi, Elena Troubitysna(✉), and György Dan

KTH – Royal Institute of Technology, Stockholm, Sweden
{poorhadi,elenatro,gyuri}@kth.se

Abstract. Modern safety-critical systems become increasingly networked and interconnected. Often the communication between the system components utilises the protocols similar to the standard Internet Protocol (IP). In particular, such protocols are used for communication between smart sensors and controller. While offering advanced capabilities such as remote diagnostics and maintenance, this also make safety-critical systems susceptible to the attacks implementable against IP-based systems. In this paper, we propose an approach to specifying a generic IP-based networked control system and formalising its security properties. We use the Event-B framework to formally analyse the impact of security attacks on safety properties of the system.

Keywords: Formal modelling · Safety-critical systems · Security · Event-B · Refinement

1 Introduction

Modern safety-critical systems become increasingly open and interconnected. In particular, the use of smart sensors and Internet of Things (IoT) enable the development of systems with advanced capabilities including remote diagnostics and proactive maintenance. Often their communication rely on standard or adapted versions of the Internet Protocol (IP). Hence, such systems become susceptible to the security attacks typical for the IP-based systems.

Networked control systems and IoT rely on remote sensing and actuation in providing their functions, including the safety-critical ones. Therefore, to ensure system dependability, we need to analyse the impact of security attacks on system safety and devise the measures for protecting the system against malicious faults.

In this paper, we propose a formal approach to modelling safety-critical networked systems and analysing the impact of the security attacks on system safety. We demonstrate how to rigorously specify the behaviour of a control system relying on a generic IP protocol for the communication with remote sensors.

Supported by organization Trafikverket.

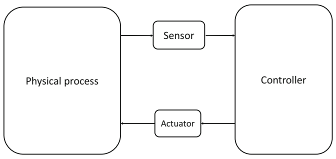


Fig. 1. Architecture of generic control system

The proposed approach supports a formal analysis of the impact of typical security attacks on the data transmitted between components. We demonstrate how to represent the results of a denial-of-service and tampering attacks by defining the corresponding system-wide invariant properties. Such an approach allows us to identify the impact that the deviations caused by the security attacks make on system safety functions.

We rely on formal modelling in Event-B [1] to systematically specify and verify both nominal and faulty system behaviour. Event-B is a rigorous approach to correct-by-construction system development by refinement. While refining the system model, we can gradually define the main stages of communication between the system components and specify the effect of security attacks.

The stepwise Event-B refinement process allows us to systematically derive the constraints and explicitly define the assumptions that should be fulfilled to guarantee system safety in the presence of attacks. The Rodin platform [2] provides an automated tool support for modelling and verification in Event-B. It automatically generates the proof obligations required for demonstrating correctness of specification and refinement and attempts to discharge them automatically. The use of an automated tool support improves scalability of formal verification and allows us to analyse the behaviour of complex networked systems. We believe that the proposed approach supports a systematic rigorous analysis of the impact of security on system safety functions.

2 A Formal Analysis of Security Properties of Networked Control Systems

2.1 Safety of a Generic Control System

In this paper, we focus on the analysis of a generic architecture of a networked control system depicted in Fig. 1. The main goal of the system is to control a certain physical process.

Without loss of generality, we assume that the state of the physical process is represented by a certain parameter s. The value of s is measured by the corresponding remote smart sensor. The sensor reading is transmitted by the communication channel connecting the sensor and controller. Based on the obtained sensor reading, the controller changes the state of the actuator, which, in its turn, affects the state of the physical process. The behaviour of the system is cyclic. At each cycle, the sensor performs the measurement of s and sends the corresponding data consisting of several packets to the controller. Based on the received data, the controller assigns a new state of the actuator. In general, the controller and actuator are also communicating over a network. However, since the behaviour of sensor-controller and controller-actuator channels are similar, we omit a detailed discussion of the latter.

Let us assume that the specification of the controller contains the operations $Setting\_Actuator\_high$ defined as follows

$$\textbf{if } input\_s \leq \ low \textbf{ then } actuator := increasing$$

and similarly $Setting\_Actuator\_low$:

$$\textbf{if } input\_s \geq \ high \textbf{ then } actuator := decreasing$$

These operations of the controller change the state of the actuator according to the obtained data from the sensor. The operations are used to achieve the desired functional behaviour of the control system.

An important non-functional requirement imposed on the system is to ensure safety, i.e., guarantee that the value of the physical parameter s does not breach certain safety threshold, i.e.,

$$s \ \in \ [low\_safe, high\_safe]$$

It is clear that the required safety property can be guaranteed only if $input\_s$ is marginally different from the actual physical value of s, i.e.,

$$|s - s\_input| \leq \delta$$

where $\delta < low - low\_safe$ (we assume that $low - low\_safe = high\_safe - high$)

Now let us investigate how the security attacks can violate these conditions.

2.2 Modelling Security Attacks and Defense Against Them

In our modelling, we aim at representing the essence of the IP communication and defining the security properties as the invariant properties over the state of the input and output buffers of the communicating parties. We analyse the sensor-controller communication.

In the IP-based systems, the components communicate with each other by exchanging packets, which are assembled into the messages by the receiver. Hence, we model the communication between the sensor and controller as a packet exchange. Lets assume that a message to be sent by a sensor consists of n packets. We assume that a packet has three parts: an integer number designating the sequence number, the header containing the required networked information, and the payload. Formally, a packet is a triple:

$$(i, h, p) \in \mathbf{N} \times Header \times Payload,$$

where *Header* and *Payload* are the abstract sets containing all possible values in the header except the sequence number and payload. The following data structure models a message:

$$message \in \{1, ..., n\} \rightarrow Header \times Payload. \tag{1}$$

For simplicity, to analyse the payloads and headers separately, we define two auxiliary functions $message_1$ and $message_2$ as follows:

$$message_1 \in \{1, ..., n\} \rightarrow Header, \quad message_2 \in \{1, ..., n\} \rightarrow Payload,$$

$$\forall\, i.\;\; 1 \leq i \leq n \;\;\Rightarrow\;\; message(i) = (message_1(i), message_2(i)).$$

We assume that at each cycle the sensor and controller should first establish a connection, i.e., open and close a session. After sending a message which initiates a new session, the sensor waits for the acknowledgement from the controller. Once the acknowledgement is received, the connection is established and the sensor starts to send the packets with the measurement data. At the receiving side, the controller stores the delivered packets in its input buffer. After the predefined number of packets have been received, it assembles them into the corresponding message by relying on the sequence number of each packet.

In our work, we focus on modelling an impact of security attacks on system safety. Hence, we need to model how an attack affects the packets received by the controller. Therefore, in general, the input buffer containing the packets to be received by the controller can be different from the output buffer containing the packets to be sent by the sensor. If a message can be represented by a total function mapping the sequence number to the packet then the input buffer can be represented by a similar but partial function.

The communication channel between the sensor and the controller is susceptible to the attacks typical for the IP- based systems. Hence, the attacks can affect both the availability and integrity of the inputs received by the controller.

Next we discuss two typical types of the attacks and formally define their effect on the controller inputs.

Packet Tampering. As a result of man-in-the-middle attack, an attacker can change the payload of some packets (the sequence number and header are unchanged). If not detected, such an attack would result in the controller making the decisions regarding the actuator state based on the incorrect input. Formally, this threat can be formalized as follows:

$$message_1(i) = delivery_1(i) \;\;\wedge\;\; delivery_2(i) \neq message_2(i). \tag{2}$$

where $1 \leq i \leq n$ and *delivery* represents the packet received by the controller.

DoS Attack. As a result of DoS attack, the receiver obtains a large number of packets initiating a new connections. Eventually, it overflows the input buffer of the controller and all the consequent packets are dropped. Formally, it can be represented as follows:

$$message_1(i) \neq delivery_1(i)$$

By defining such system-wide properties, we can formally specify the impact of a security attack on the system. Hence, we obtain a formal ground for identifying the impact of the security control mechanisms as well as the effect of a security attack on safety.

Let us specify a behaviour of such widely-used security control mechanisms as a security gateway. We introduce the *Detection* function that maps each packet to a boolean value:

$$Detection : \{1, ..., n\} \times H \times P \rightarrow BOOL \tag{3}$$

Mapping to the value $TRUE$ denotes that an attack has been detected.

Therefore, we can ensure safety in the presence of an active attacker who only tampers the payload of some packets if and only if we can prove the following security properties.

Suppose that the security gateway receives the ith packet, then the following property should hold.

$$\begin{aligned}&detection(i, delivery_1(i), delivery_2(i)) = TRUE \;\Leftrightarrow\\ &message_1(i) = deliveryr_1(i) \quad \wedge \quad message_2(i) \neq delivery_2(i).\end{aligned}$$

Obviously, once the attack is detected, the controller can no longer rely on the data received from the sensor. Hence, to ensure that the system does not breach safety when an attack is detected, the specification of the controller should contain some fall-back operations. Such an operation can be, e.g., the use of the last good value received by the controller. Such a mechanism would allow the system to continue to function for a few cycles and might alleviate the impact of a security attack in case it had a short duration. However, if the attack persists the system should be shut down. Alternatively, the controller can directly put the system in a safe but non-operational state upon detection of a security attack.

In the next section, we give a brief overview of our formal modelling framework Event-B and then demonstrate how to formally specify a networked control system and its security properties in Event-B.

3 Event-B

Event-B is a state-based formal approach that promotes the correct-by-construction development paradigm and formal verification by theorem proving [1]. In Event-B, a system model is specified as an *abstract state machine*. An abstract state machine encapsulates the model state, represented as a collection of variables, and defines operations on the state, i.e., it describes the dynamic behaviour of a modelled system. The variables are strongly typed by the constraining predicates that, together with other important system properties, are defined as model *invariants*. Usually, a machine has an accompanying component, called a *context*, which includes user-defined sets, constants and their properties given as a list of model axioms.

The dynamic behaviour of the system is defined by a collection of atomic *events*. Generally, an event has the following form:

$$e \;\widehat{=}\; \textbf{any}\ a\ \textbf{where}\ G_e\ \textbf{then}\ R_e\ \textbf{end},$$

where e is the event's name, a is the list of local variables, G_e is the event *guard*), and R_e is the event action.

The guard is a state predicate that defines the conditions under which the action can be executed, i.e., when the event is *enabled.* If several events are enabled at the same time, any of them can be chosen for execution non-deterministically. If none of the events is enabled then the system deadlocks. The occurrence of events represents the observable behaviour of the system.

In general, the action of an event is a parallel composition of deterministic or non-deterministic assignments. In Event-B, this assignment is semantically defined as the next-state relation R_e. A deterministic assignment, $x := E(x, y)$, has the standard syntax and meaning. A non-deterministic assignment is denoted either as $x :\in S$, where S is a set of values, or $x :\mid P(x, y, x')$, where P is a predicate relating initial values of x, y to some final value of x'. As a result of such a non-deterministic assignment, x can get any value belonging to S or according to P.

Event-B employs a top-down refinement-based approach to system development. A development starts from an abstract specification that nondeterministically models most essential functional requirements. In a sequence of refinement steps, we gradually reduce nondeterminism and introduce detailed design decisions. The consistency of Event-B models, i.e., verification of well-formedness, invariant preservation as well as correctness of refinement steps, is demonstrated by proving the relevant verification theorems – proof obligations.

Proof obligations are expressed as logical sequences, ensuring that the transformation is performed in a correctness-preserving way. For instance, *invariant preservation* property for the given model invariant I_j is formulated as follows:

$$A(d, c),\ I_j(d, c, v),\ G_e(d, c, a, v),\ R_e(d, c, a, v, v') \;\vdash\; I_j(d, c, v') \qquad \text{(INV)}$$

where A are model axioms, G_e is the event guard, d are model sets, c are model constants, a are the event local variables and v, v' are the variable values before and after the event execution.

Modelling, refinement and verification in Event-B is supported by an automated tool – Rodin platform [2]. The platform provides the designers with an integrated modelling environment, supports automatic generation and proving of the necessary proof obligations. Moreover, various plug-ins created for Rodin platform allow a modeller to transform models from one representation to another. They also give access to various verification engines (theorem provers, model checkers, SMT solvers).

4 Formal Development of a Safety-Critical System with Security Consideration

In this section, we demonstrate how to formally model a communication between the sensor and controller in the presence of tampering and DoS attacks. We create a formal model of a packet tampering and DoS attack and the introduce a defense mechanism ensuring that safety can be maintained when the system is attacked.

In our model, a control cycle starts from an attempt to establish a connection between the sensor and the controller and finishes with the connection termination either with successfully completed transmission or aborted transmission due to the detected security attack.

We start by explaining how the message transfer is modelled. Since we consider an IP-based systems, to establish a connection, the sender – a smart sensor – first sends a session invitation message to the receiver – the controller. The controller replies with an acknowledgement and opens a connection. To enable modelling of a security attack, we introduce a modelling abstraction – an intermediate buffer. The intermediate buffer models a behaviour of a transmission channel. When the channel is not attacked then a packet transmitted by a receiver is stored in the intermediate buffer unchanged and consequently copied to the receiver's input buffer.

When the system is under a tampering attack, the intermediate buffer allows us to model an effect of an attack – the payload of the packet is changed in the intermediate buffer. Consequently, the receiver obtains a packet, which is different from the packet, which was sent by the sender. To model DoS attack, we use the intermediate buffer to insert packets that have never been sent by the sender. Let us observe, that such a modelling approach can also be easily adapted to model a replay attack. We introduce a similar buffer to model a communication in the reverse direction. The similar buffer is introduced to store the acknowledgements sent by the controller to the sensor.

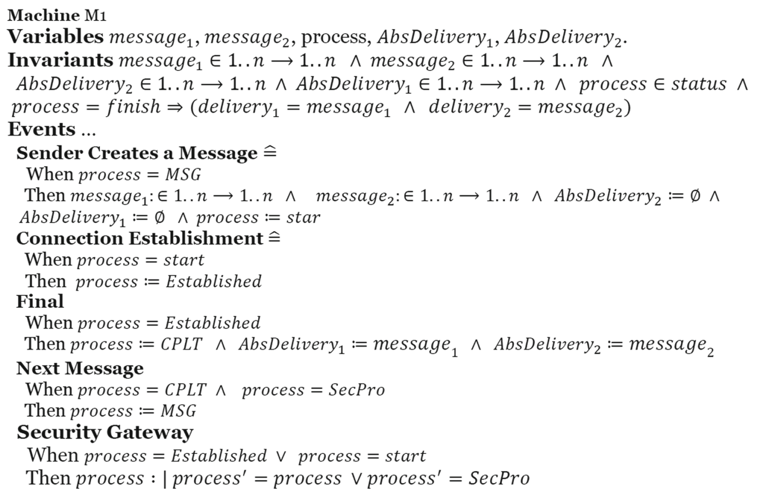
We model this cyclic behavior as a sequence of phases.

$$... \; MSG \Rightarrow Start \Rightarrow Established \Rightarrow CPLT \; or \; SecPro \Rightarrow MSG \; ....$$

In our abstract specification, outlined in Fig. 2 the sender and receiver share the state space. Hence, the successful transmission of a packet can be represented as a simple assignment of the buffer of the sender to the buffer of the receiver.

In the abstract specification, a variable *process* models the different phases of communication.

In the phase MSG, the sensors generates a message *message* to be sent. In phase *Start*, the transmission process begins. When *process* is equal to *Established*, the sensor and controller are exchanging packets. In the abstract specification, the controller receives the whole message at once. At the end of this phase, *process* can be in phases $CPLT$ if an attack is detected or in *SecPro* otherwise.



Machine M1
Variables $message_1, message_2$, process, $AbsDelivery_1, AbsDelivery_2$.
Invariants $message_1 \in 1..n \longrightarrow 1..n \ \wedge \ message_2 \in 1..n \longrightarrow 1..n \ \wedge$
$AbsDelivery_2 \in 1..n \longrightarrow 1..n \ \wedge \ AbsDelivery_1 \in 1..n \longrightarrow 1..n \ \wedge \ process \in status \ \wedge$
$process = finish \Rightarrow (delivery_1 = message_1 \ \wedge \ delivery_2 = message_2)$
Events ...
Sender Creates a Message $\widehat{=}$
When $process = MSG$
Then $message_1 :\in 1..n \longrightarrow 1..n \ \wedge \ message_2 :\in 1..n \longrightarrow 1..n \ \wedge \ AbsDelivery_2 := \emptyset \ \wedge$
$AbsDelivery_1 := \emptyset \ \wedge \ process := star$
Connection Establishment $\widehat{=}$
When $process = start$
Then $process := Established$
Final
When $process = Established$
Then $process := CPLT \ \wedge \ AbsDelivery_1 := message_1 \ \wedge \ AbsDelivery_2 := message_2$
Next Message
When $process = CPLT \ \wedge \ process = SecPro$
Then $process := MSG$
Security Gateway
When $process = Established \ \vee \ process = start$
Then $process : | \ process' = process \ \vee process' = SecPro$

Fig. 2. The structure of abstract specification

In this level, we can prove the following invariant to show the message is sent successfully:

$$process = CPLT \Rightarrow AbsDelivery = message.$$

The first refinement step aims at decomposing a message into a number of packets and modelling their step-by-step transmission. We introduce a the new event *receive*, which models receiving a packet by the controller. The controller keeps track of the received packets and after all the packets have been received, composes them into a message. We define the following invariant to model the fact that for a message to be successfully received all its packets should be delivered successfully.

$$delivery = (1..(c-1)) \lhd message$$

where $\lhd$ stands for a domain restriction.

Our next refinement step focuses on separating state spaces of the sender – the sensor – and receiver – the controller and introducing an intermediate buffer between them to model the affect of an attack. However, at this level, the sensor can still access the state of the controller, i.e., our security properties are yet not defined in an entirely distributed way.

The intermediate buffer is modeled by two variables denoted by $BufCounter$ and $BufData$.

$$BufCounter \in (0..n) \ \wedge \ BufData \in Header \times Payload.$$

We define a new event *send* to add (i.e., assign) a new packet to the intermediate buffer. Correspondingly, we refine the event *receive* to model the fact that the packets received by the controller are transmitted via a communication channel, i.e., are first stored (an modified, in case of an attack) in the intermediate buffer.

In the previous refinement step, the sensor could still read the variable of the controller. In this specification, we remove this modelling abstraction. Namely, we model the behaviour of the sensor waiting for an acknowledgment before starting to send a new packet.

To model this, we define a variable *SensorRcv* of type Boolean. It specifies the conditions defining whether the sensor should send a new packet, i.e., has received the acknowledgement for the previously sent packet. We also introduce the intermediate buffer *AckCh* for the controller-sensor communication and two new events modelling sending and receiving the acknowledgments. When the controller receives a packet, it changes the value of *AckCh* to indicate that the previous packet has been delivered successfully.

At this point of the formal development, we have completed modeling the communication between the sender – the smart sensor – and a receiver – the controller. The system model is distributed, i.e., the state spaces of the communicating components are disjoint. All the invariant properties are defined over the distributed state space of the system. Now we are ready to model an effect of an attack on the system behaviour.

In the fourth refinement step, we model the attacker's behavior and security control mechanisms. To achieve this, we introduce the events tampering and injection defined as follows:

$$\begin{array}{l}\textbf{tampering} \mathrel{\widehat{=}} \\ \qquad \textbf{when}\ process = Established\ \wedge\ BufCounter = c \\ \qquad \textbf{then}\ CBufData_2 :\in Payload.\end{array}$$

$$\begin{array}{l}\textbf{injection} \mathrel{\widehat{=}} \\ \qquad \textbf{when}\ process = Established\ \wedge\ BufCounter = c \\ \qquad \textbf{then}\ BufData2_2 :\in Payload\ \wedge\ BufData2_1 :\in Header \\ \qquad\qquad \wedge\ BufCounter2 := c.\end{array}$$

The events become enabled after the sensor sends a new packet. The **tampering** event results in changing the payload of the packet. The payload is changed to any arbitrary value in the set *Payload.* The **injection** event results in generating an new packet that is stored in the intermediate buffer.

To model a security control mechanism, we introduce a variable $validity \in \{Checked, Nchecked\}$. The variable is modified by the controller. It models the outcome of integrity verification for the last packet stored in *delivery*, i.e., represent the fact that the packet has either passed the security verification or not. Whenever a new packet arrives to the controller side, the controller verifies its integrity, which is abstractly modelled by an event **gateway**, which assigns a new value to the variable *validity*. If the packet is valid then the variable *validity* receives the value *Checked* and the controller sends the corresponding acknowledgement to the sensor. If the verification fails then the system terminates the connection and *process* becomes *SecPro*.

Now we can prove the general security property defined in Sect. 2.

$$\begin{aligned}&validity = Checked \Rightarrow\\ &detection(c-1, delivery_1(c-1), delivery_2(c-1)) = TRUE \;\Leftrightarrow\\ &message_1(c-1) = delivery_1(c-1) \quad \wedge \quad message_2(c-1) \neq delivery_2(c-1).\end{aligned}$$

The introduction of the security protection mechanism – the secure gateway – allows us to ensure that the tampered or injected messages would not be accepted by the controller as an input. Hence, we can guarantee that the controller input would remain sufficiently close to the real physical value of the controlled parameter. Otherwise, if the secure gateway does establish message validity, i.e., the message is suspected to be tampered or injected, the controller can rely on its own fault tolerance mechanisms to ensure safety.

4.1 Discussion of Development

While modelling, we have adopted an implicit discrete model of time. Namely, we define the abstract function representing the change in the dynamics of the controlled process as well as the constraints relating the components behaviour in the successive iterations. Such an approach is based on our previous experience in modelling control systems, e.g., [11]. Such an approach allows us to define system invariant properties in relation to a particular phase of control loop execution or a communication progress. An alternative way to approach the problem of modelling time could be to rely on real-time extension of Event-B [17]. In such a way, we could also express the time-related properties of data transmission as well as define time explicitly the time-stamps of the packets.

Another abstraction, in which we relied in our modelling, is a representation of a networked architecture. In the proposed chain of refinements, we have gradually moved from modelling a centralised architecture to separating state space of communicating components. However, formally, the behaviour of the components is modelled within a single monolithic specification. To address this issue and explicitly represent each component separately, we can rely on the modularisation approach [12–14], which supports compositional reasoning and specification patterns [15].

5 Related Work and Conclusions

The problem of safety and security interactions has recently received a significant research attention. It has been recognised that there is a clear need for the approaches facilitating an integrated analysis of safety and security [4,8,9].

This issue has been addressed by several techniques demonstrating how to adapt traditional safety techniques like FMECA and fault trees to perform a security-informed safety analysis [5,8]. The techniques aim at providing the engineers with a structured way to discover and analyse security vulnerabilities that have safety implications. Since the use of such techniques facilitate a systematic analysis of failure modes and results in discovering important safety and security requirements, the proposed approaches provide a valuable input for our modelling.

There are several works that address formal analysis of safety and security requirements interactions [6,10]. Majority of these works demonstrate how to find conflicts between them. A typical scenario used to demonstrate this is a contradiction between the access control rules and safety measure. In our approach, we treat the problem of safety-security interplay at a more detailed level. Namely, we model the data transmission in an IP-based system and demonstrate how a security attack affects system behaviour on the level of packet transmission and as a result can jeopardise safety.

The distributed MILS approach [3,7] employs a number of advanced modelling techniques to create a platform for a formal architectural analysis of safety and security. The approach supports a powerful analysis of the properties of the data flow using model checking and facilitates derivation of security contracts. Since our approach enables incremental construction of complex distributed architectures, it would be interesting to combine these techniques to support an integrated safety-security analysis throughout the entire formal model-based system development.

An explicit reasoning about communication between decentralised components in Event-B has been discussed in [16]. In the similar way, the behaviour of components and data is represented via the corresponding buffers. However, this work does not consider an effect of security attacks on the transferred packets and focuses on another communication protocol.

A formal development of safety-security interplay has been carried out in a number of recent works [18–21]. In these approaches, a more high-level analysis of security impact has been undertaken. These works focus on modelling data-flow related properties as well as integration of different safety analysis techniques to identify the impact of security attacks on safety. Despite of sharing many common modelling solutions, in this paper, we focused on a different aspect – modelling of an IP-based system and analysis of the impact of the typical IP-related attacks on safety.

In this paper, we have presented a formal approach to modelling a security attacks in an IP-based system and their impact of safety. Our approach considers the detailed data transmission process between the sensor and the controller. It allowed us to explicitly model the actions of the attackers and their impact on the transmitted messages. As a result, we were able to formally demonstrate that an introduction of a security control mechanism allows us to guarantee preservation of safety.

As a future work, we are planning to continue to study different types of attacks at a detailed level and validate our approach by large-scale case studies.

References

1. Abrial, J.-R.: Modeling in Event-B. Cambridge University Press, Cambridge (2010)
2. Rodin: Event-B platform. http://www.event-b.org

3. Bytschkow, D., Quilbeuf, J., Igna, G., Ruess, H.: Distributed MILS architectural approach for secure smart grids. In: Cuellar, J. (ed.) SmartGridSec 2014. LNCS, vol. 8448, pp. 16–29. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10329-7_2
4. Young, W., Leveson, N.G.: An integrated approach to safety and security based on systems theory. Commun. ACM **57—-2**, 31–35 (2014)
5. Fovino, I.N., Masera, M., De Cian, A.: Integrating cyber attacks within fault trees. Rel. Eng. Syst. Saf. **94—-9**, 1394–1402 (2009)
6. Kriaa, S., Bouissou, M., Colin, F., Halgand, Y., Pietre-Cambacedes, L.: Safety and security interactions modeling using the BDMP formalism: case study of a pipeline. In: Bondavalli, A., Di Giandomenico, F. (eds.) SAFECOMP 2014. LNCS, vol. 8666, pp. 326–341. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10506-2_22
7. Cimatti, A., DeLong, R., Marcantonio, D., Tonetta, S.: Combining MILS with contract-based design for safety and security requirements. In: Koornneef, F., van Gulijk, C. (eds.) SAFECOMP 2015. LNCS, vol. 9338, pp. 264–276. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-24249-1_23
8. Schmittner, C., Gruber, T., Puschner, P., Schoitsch, E.: Security application of failure mode and effect analysis (FMEA). In: Bondavalli, A., Di Giandomenico, F. (eds.) SAFECOMP 2014. LNCS, vol. 8666, pp. 310–325. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10506-2_21
9. Steiner, M., Liggesmeyer, P.: Combination of safety and security analysis - finding security problems that threaten the safety of a system. In: SAFECOMP 2013 - Workshop DECS-2013, HAL (2013)
10. Troubitsyna, E., Laibinis, L., Pereverzeva, I., Kuismin, T., Ilic, D., Latvala, T.: Towards security-explicit formal modelling of safety-critical systems. In: Skavhaug, A., Guiochet, J., Bitsch, F. (eds.) SAFECOMP 2016. LNCS, vol. 9922, pp. 213–225. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-45477-1_17
11. Laibinis, L., Troubitsyna, E.: Refinement of fault tolerant control systems in B. In: Heisel, M., Liggesmeyer, P., Wittmann, S. (eds.) SAFECOMP 2004. LNCS, vol. 3219, pp. 254–268. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30138-7_22
12. Iliasov, A., et al.: Supporting reuse in event B development: modularisation approach. In: Frappier, M., Glässer, U., Khurshid, S., Laleau, R., Reeves, S. (eds.) ABZ 2010. LNCS, vol. 5977, pp. 174–188. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-11811-1_14
13. Iliasov, A., et al.: Developing mode-rich satellite software by refinement in Event-B. Sci. Comput. Program. **18**(7), 884–905 (2013)
14. Iliasov, A., Troubitsyna, E., Laibinis, L., Romanovsky, A., Varpaaniemi, K., Väisänen, P., Ilic, D., Latvala, T.: Verifying mode consistency for on-board satellite software. In: Schoitsch, E. (ed.) SAFECOMP 2010. LNCS, vol. 6351, pp. 126–141. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15651-9_10
15. Iliasov, A., Troubitsyna, E., Laibinis, L., Romanovsky, A.: Patterns for refinement automation. In: de Boer, F.S., Bonsangue, M.M., Hallerstede, S., Leuschel, M. (eds.) FMCO 2009. LNCS, vol. 6286, pp. 70–88. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-17071-3_4
16. Iliasov, A., Laibinis, L., Troubitsyna, E., Romanovsky, A.: Formal derivation of a distributed program in event B. In: Qin, S., Qiu, Z. (eds.) ICFEM 2011. LNCS, vol. 6991, pp. 420–436. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-24559-6_29

17. Iliasov, A., Romanovsky, A., Laibinis, L., Troubitsyna, E., Latvala, T.: Augmenting Event-B modelling with real-time verification. In: FormSERA 2012, pp. 51–57. IEEE (2012)
18. Vistbakka, I., Troubitsyna, E., Kuismin, T., Latvala, T.: Co-engineering safety and security in industrial control systems: a formal outlook. In: Romanovsky, A., Troubitsyna, E.A. (eds.) SERENE 2017. LNCS, vol. 10479, pp. 96–114. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-65948-0_7
19. Vistbakka, I., Troubitsyna, E.: Towards a formal approach to analysing security of safety-critical systems. In: EDCC 2018, pp. 182–189. IEEE (2018)
20. Troubitsyna, E., Vistbakka, I.: Deriving and formalising safety and security requirements for control systems. In: Gallina, B., Skavhaug, A., Bitsch, F. (eds.) SAFECOMP 2018. LNCS, vol. 11093, pp. 107–122. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-99130-6_8
21. Vistbakka, I., Troubitsyna, E.: Pattern-based formal approach to analyse security and safety of control systems. In: Papadopoulos, Y., Aslansefat, K., Katsaros, P., Bozzano, M. (eds.) IMBSA 2019. LNCS, vol. 11842, pp. 363–378. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-32872-6_24