Towards Security Case Run-time Adaptation by System Decomposition into Services

Elena Lisova and Aida Čaušević
Mälardalen Real-Time Research Centre, Mälardalen University,
Västerås, Sweden
{elena.lisova, aida.causevic}@mdh.se

***Abstract*—For interconnected and complex systems, security is paramount for establishing trust in their correctness and design adequacy. Thus, security needs to be assured and a corresponding security assurance case needs to be presented to system stakeholders, security assessors, as well as to system users. However, security is dynamic by its nature and to maintain its acceptable security level, frequent updates might be required. Traditionally, a security assurance case is built from scratch whenever a change occurs, however given the cost of resources needed for such a task, a more effective and less time consuming way of handling updates is needed. Hence, the challenge of security case run-time adaptation is considered in this work. We survey the state of the art in security assurance and security case development to refine the challenge and identify system decomposition as one the enablers for security case run-time adaptation. We propose to apply system decomposition in terms of services and use service choreographies to facilitate security case run-time adaptation. The proposed approach is illustrated on an E-gas example.**

I. Introduction

For today's system with increasing complexity and a large number of interconnections with other systems, security becomes a necessary property that needs to be not only provided but also structured and argued in a clear way. It is especially the case when humans are involved in the loop and there are direct physical interactions between humans and systems, as such a setting makes the system safety-critical. For instance, an autonomous vehicle, working in cooperation with humans or just near-by humans, needs to be acceptably secure, i.e., there should be enough confidence that its security level is adequate and in line with the current state of the art in terms of threats, attacks and system vulnerabilities.

A systematic way to build an acceptable level of confidence in system security is to collect evidences and arguments over security measures adequacy in a *security case* [1]. Providing a security case is a demanding task, it requires a significant amount of resources in terms of time, cost, and a human effort. An analogy can be made with a *safety case* [2], which is required to be built for certification purposes. The cost of certification for a safety-critical system is estimated up to 75 % of its development costs [3]. However, security solutions are much more dynamic compared to the safety ones with respect to continuous updates and further refinements. More importantly, a system cannot be stamped as being "*secure*" once and forever. Security as a system property can be provided and guaranteed to a specified extent only for a current state of the art, as new system vulnerabilities are constantly exposed and new attack techniques are continuously being developed [4]. Given security being dynamic by its nature it requires run-time updates and refinements. Since developing a security case from scratch whenever an update occurs, is not feasible, the challenge of handling updates within a security case needs to be addressed. The notion of a dynamic assurance case has already been proposed in the safety domain [5], e.g., the introduction of a set of rules for updates and a set of monitors for establishing a link between the system and a confidence structure within the safety case. We believe that such techniques can be adopted, further developed and complemented with relevant solutions to handle a dynamic security assurance case.

Enabling security case run-time adaptation facilitates trust assurance, addresses a challenge of run-time certification [6] and assurance of adaptive systems [7]. Developing a dynamic security case will allow adaptation of the case at run-time to cater for emerging threats and vulnerabilities, as well data from monitoring of the executing system. Given this, it is possible to understand ways to identify parts of the case that are affected by a particular change and re-examine them. This enables a security case adaptable at run-time and possibly applicable also in security-aware safety cases (i.e., safety cases where safety relevant security aspects are considered).

In order to facilitate security case run-time adaptation one has to find a way to decompose a system and develop an approach to trace dependencies between an introduced change in a security solution and arguments presented in the corresponding security case. We have identified a paradigm of services, i.e., autonomous, platform-independent entities that can be described, published, discovered, and loosely coupled [8], as a solution to enable such a decomposition. Service interfaces carry information regarding the basic service functionality (i.e., pre-, and postcondition for a service to be executed), publishing, finding, and binding of services (i.e., a service level agreement (SLA)). Furthermore, we see the benefit of using the notion of service choreographies (i.e., a distributed form of service composition where services follow a set of rules, and follow their expected roles, according to the behavior of the other participating services) in order to enable security case run-time adaptation.

The main contribution of this paper is the introduction of initial constructs for the security case run-time adaptation. Such an approach is complex by its nature, therefore we describe its further refinement into sub-challenges based on the current state of the art for security assurance and security case development. Further, the identified sub-challenge of system decomposition is addressed by proposing to trace dependencies between an introduced system update and corresponding security case arguments and evidences that need to be revised based on system decomposition into services. We illustrate our idea using an E-gas example.

The reminder of the paper is structured as follows: Section II presents necessary background related to the assurance case and service paradigm. Next, the state of the art in security assurance and case is investigated in Section III. In Section IV we present the approach for mapping an update with relevant security case parts based on service choreographies. The example of E-gas, used to illustrate our approach, is introduced in Section V. Finally, conclusions with future work directions are presented in Sections VI.

II. Background

In this section we present necessary background related to the assurance case and the service paradigm.

A. Assurance Case

An assurance case can be defined as "*an enabling mechanism to show that the system meets its prioritized requirements*" [9], with the aim to reason about system trustworthiness. It can be built for a system property such as safety, security or ethics and thus become safety [10], security [1] or ethics assurance case [11]. The rationale behind a case is to identify a high level claim related to a chosen system property, collect and present arguments supporting the claim, and associate it with corresponding evidences. It can support both quantitative and qualitative forms of analysis as an evaluation criteria [12]. It can be described using formal methods or semi-formal methods. Claims are assigned with an assurance level which can be associated to a probability or a risk level. There are two main approaches to represent arguments: Goal Structuring Notation (GSN) [13] and Claims, Arguments and Evidence (CAE) [14]. GSN provides arguments by showing how goals, i.e., claims, can be decomposed into subgoals and supported by solutions, i.e., evidences. In GSN, goals are related to a specific context in which they are valid and connection between goals and solutions are presented via arguments. The CAE approach has a very similar structure without implicitly separating context of claims.

B. Common Criteria

The Common Criteria for Information Technology Security Evaluation (CC) [15] is an international standard developed by ISO/IEC 15408 for computer security evaluation. The CC process starts with protection profile (PP) evaluation, in which PP describes a type of target of evaluation (TOE) by a set of relevant requirements. The next step is security target (ST) evaluation, where ST describes a specific TOE including implementation specifics. The last part is TOE evaluation. CC distinguishes seven evaluation levels (EALs), where each level has a set of techniques for evaluation and the higher EAL is, the more formal techniques are. In practice, achieving EAL above EAL4 is hardly feasible for complex systems.

C. Services and Service Compositions

A *software service* is a set of functions provided by a server software or a system to a client software or a system, usually accessible through an application programming interface [16]. Services can be created, invoked, composed and destroyed on demand. They are assumed to be platform independent and applicable to heterogeneous applications. A service composition allows development of composite services with the main goal of a reusable functionality being provided by existing services in a low cost and rapid development process on demand. One of the fundamental characteristics of services is separation of interfaces from the service behavioral description. A publicly available service interface information specifies service properties such as a service type, capacity, time-to-serve, etc. It is visible to service users and used to find and invoke services most suitable for their needs. On the other hand, internal behavior-related information such as functionality representation is hidden from the service user and available only to service developers. The paradigm of developing a system as a set of services may be seen as a cost-efficient development by reusing functionalities from available services. Also, a service becomes a single point of maintenance for a common functionality.

One of the main principles related to services is the idea of composing services by discovering and invoking them on demand, rather than building the whole application from scratch, at design time. A service composition can be achieved either through *orchestration* or *choreography*. The former assumes the existence of a central controller responsible for scheduling service execution according to the user demands, while the latter assumes a mechanism of message exchange between participants in the composition, without requiring a central coordinator. Choreographies rely on the service-to-service communication style, meaning that each service in choreography/composition has knowledge when to execute its operations and with whom to interact. Communication between services is established by defining a multi-party protocol, that enables accomplishment of the overall choreography objectives in a fully distributed way.

For a service to be formally deployed and executed a service specification that involves defining contractual agreements, i.e., SLAs, between the service provider and its user, has to exist. Given SLAs, service users are able to establish trust in that the service outcome is what they have demanded during the service negotiation process.

III. The state of the art

The current state of the art in terms of security assurance and particularly security case development is investigated in this section.

A. Security Assurance

Security is a system property that needs to be assured and clearly communicated to both system stakeholders and its users. Kizza defines security assurance as "*a continuous security state of the security process*" [17] and specifies the following steps of the security process: a system security policy, a security requirements specification, a threat identification, a threat analysis, a vulnerability identification and assessment, a security certification, a security monitoring and auditing.

Considering assurance as a process aligned with the security process, Agudo et al. [18] argue that security assurance needs to be considered during the system development process as different phases of development may require different levels of assurance. In this way an assurance case as an outcome of a security assurance is created during the system development process. Chen et al. [7] also point out continuous assurance through the entire lifecycle as a way of enabling run-time assurance of self-adaptive systems. They also present a set of assurance techniques and their classification.

A security assurance during the system lifecycle can be composed from techniques applied at different phases. For example requirements elicitation, a step during concept phase of system development, has been addressed by Nam et al. [19] presenting a modeling framework to elicit and validate requirements to support security assurance. They propose a work-flow for incremental security assurance and provide a tool design with two domain specific languages, namely Alisa and Assure.

As described in Section II, CC is a broadly used standard for security evaluation. Thus, there are many works targeting to align security process with CC especially for requirements elicitation [20], [21]. CC is one of the most commonly used standards, however it evaluates design methods, not security functionality and it does not support adaptation. CC can be used as a part of argumentation over adequacy of system security solutions design. For example, Akram et al. [22] present a framework for security assurance based on CC, where they propose a mechanism that on demand can provide assurance and validation of an implemented security solution.

There are several publications that introduce security assurance evaluation used to argue over a certain system security level. Hecker and Riguidel [23] argue over necessity to assure that networked IT systems have certain security properties and to which degree those properties are guaranteed. They propose two approaches for security assurance evaluation, namely a spec-based and a direct system security assurance evaluation. The first implies creating a security specification and assigning an assurance level similarly to how it is done in CC. Levels are hierarchical and the authors specify how to evaluate assurance. The second approach includes a manager that judges the current situation of security assurance. Direct evaluation produces results for a particular environment and evaluation results for the same system can vary in different environments. That can be considered as a disadvantage or as a sign that security assurance should be context dependent.

An approach for security assurance evaluation performed after risk assessment and countermeasures deployment is proposed by Ouedrago at el. [24]. The authors connect an assurance level of the system with its criticality, i.e., evaluation of confidence in security measures adequacy depends on a system criticality level. The evaluation methodology is a multi-step process that depends to some extent on CC criteria when choosing a suitable metrics to evaluate system components, and addresses the challenge of security decomposition by defining an aggregation function. The function produces a security measure assurance level and takes as input availability, conformity and context security criticality. The evaluation methodology consists of the following steps: (*i*) modeling of system components relevant for assurance; (*ii*) identifying metrics for checking all relevant components; (*iii*) components evaluation based on the identified metrics; (*iv*) an assurance level aggregation. The metrics from the second step vary depending on component criticality and can be characterized based on coverage, depth and rigorousness [25], inspired by CC criteria. The last presented step enables security decomposition. For doing so, authors define an aggregation function that produces a security measure assurance level and takes as input availability, conformity and context security criticality. The challenge is to aggregate security measure assurance levels and provide a system security level. Authors also stress importance of the context and relating system criticality level with a particular context.

One of the challenges in security assurance is claiming a system security level based on evaluation of separate security mechanisms, i.e., the challenge of security not being decomposable. To this end, security is similar to safety, which is not decomposable, meaning that a sufficiently safe system component does not imply anything in regards to system safety, or that two proved to be sufficiently safe components do not imply that their combination is also sufficiently safe. The same applies to security, e.g., if data is protected within a system component and two of such components are composed to work together, communication between components needs to be investigated as well for its vulnerabilities. Pierce and Rapids [26] discuss challenges of system level security and vulnerability analysis decomposition at the example of integrated modular avionics architectures. Formoso and Felici [27] present an evidence-based approach for security and privacy assurance based on a control-evidence model. Similarly, given that a system of systems is defined in terms of services, it can be analyzed through established SLAs between its constituent systems. Rak [28] presents security assurance of a cloud application via SLA decomposition. A system representation via services allows derivation of connections between services through their composition, i.e., when there is a change related to a particular service all other services potentially affected by it can be defined as ones belonging to a branch of services with this service as a root. Thus, it can enable mapping of an update to a system composition.

In order to facilitate an assurance run-time adaptation, a system component assurance and its relation to system assurance needs to be investigated. Rauf and Troubitsyna [29] present a model-driven framework for open source software component assurance aiming to automate the process of security assurance in presence of frequent updates. The framework is based on a "*design by contract*" approach [30]. The approach implies derivation of contracts based on the system design and its follow-up verification by checking those contracts. A wrapper provided with contracts and requirements is used on top of open source software for its assurance. A way to decompose system at a logical level is identifying services it is composed of. Montenegro et al. [31] discuss service security assurance and propose a concept of a certificate profile that captures means of evaluation, a certification domain, and vocabulary to express security aspects.

Security assurance is the main driver for our work, and given that a system decomposition into services seems as a possible approach for enabling assurance at run-time, we decide to look into service-based safety assurance proposed by Harris et al. [32] to investigate whether it is applicable to the security domain. The authors argue applicability of a service based approach for safety assurance, as it reflects system cooperation within a larger system of systems. The authors also introduce a wrapper as a technique supporting services assurance and introduce a term "*safety service wrapper*". To address handling of a change, a change level is introduced to determine what is acceptable for a service as a change. They use modular assurance techniques with assurance contracts as a solution for service based assurance. The authors also claim a need for security informed safety services, thus services seem to be a possible ground to synchronize safety and security cases development. However, to build an assurance case for a system decomposed into services, service assurance arguments need to be established and adjusted for the property that is targeted by assurance. A promising idea is to investigate applicability of the "*wrapper*" concept for security and to which degree safety and security are overlapped on the service level.

B. Security Case

A security case is a way of presenting security assurance as a collection of security claims, arguments, and evidences. Netkachova et al. [33] present a structured security informed reliability assurance case based on CAE. The authors use a preliminary interdependency analysis to model and evaluate system properties. Such an analysis can help to map an update with its affected area in the system. Bloomfield et al. [34] present one more work based on CAE investigating how a CAE based assurance framework can be used to generate security policies. This work has been performed together with a safety regulator, and based on regulator's vision of service security and safety the authors have decomposed the policies further using CAE to a set of cyber-security related objectives. To address dynamic nature of security, the authors decompose objectives based on time and propose claims about present and future. Considering GSN, Saruwatari and Yamamoto [35] propose an extension of GSN for security assurance by introducing a concept of an actor. Such concept facilitates consideration of "*intent*", i.e., turning safety argumentation towards including security considerations.

Within a security case, arguments connect evidences with claims. Rodes et al. [36] investigate security arguments by facilitating security metrics that need to be complete and valid, and propose a framework for argument assessment that generates and interprets security metrics on the example of software systems. Security is quantified within the framework in terms of a level of beliefs, i.e., a confidence level of arguments. The framework is developed within a Metric Based Security Argument approach. The approach rationale consists of stating the claim, developing arguments supporting it and identifying metrics that assess argument confidence. The concept of confidence in argument assessment is similar to the concept of Assurance Claim Points introduced by Graydon [37]. Rodes et al. provide a definition of the term confidence that includes appropriateness, sufficiency, and trustworthiness. Run-time assessment of confidence in arguments can facilitate run-time adaptation of a security case, however, within the proposed framework the assessment is done off-line by experts.

Rodes et al. [38] conduct a case study demonstrating security case development on an example of the S^3 approach. S^3 is a security technique counteracting injection commands [39]. A particular security technique has been in the focus of this work, however the idea is to apply the method of security case development to the whole IT system. Within case development, the authors consider factors that this technique can influence including new vulnerabilities, which implies an impact analysis from a security point of view. The authors propose a four levels structure of the security argument: the first level where system assets and security requirements are organized; the second level includes arguments over vulnerabilities related to each requirement and grouping these vulnerabilities into attack classes; the next level is responsible for further decomposition of attacks classes; finally the fourth level presents arguments over mitigation techniques for each vulnerability. The idea is that arguments for security case differ compared to the ones used in safety cases due to security specifics and thus an arguments patterns extension is needed.

IV. Mapping an Update to a Security Case

Given the findings presented in Section III, we can conclude that there is a need to introduce a structured way in providing and maintaining a security case. Further we discuss the challenge of run-time adaptation and propose an approach for mapping a security update with corresponding parts of the system security assurance case that needs to be re-examined.

A. The Challenge of Security Case Run-time Adaptation

As the main reason for security assurance cases not being a widely accepted concept in practice we recognize the fact that its static structure requires (re-)building the case from scratch given any update at run-time. Enabling security run-time adaptation is the main challenge for assuring system security, especially in cases of systems built from other systems. Run-time adaptation would allow handling updates in a cost-efficient and effective way as well as facilitate continuous security assurance of systems that adapt at run-time. Based on the state of the art in security assurance and security cases presented in Section III we divide the formulated challenge in the following sub-challenges: i) **system decomposition** with the purpose of enabling identification of systems parts affected by a the given update; ii) **a security case structure**, including argument patterns and respective evidence handling, reflecting applied system decomposition; iii) **run-time assessment of confidence in arguments**, i.e., given a change in the system an impact analysis would be required to evaluate how the implemented update has affected existing arguments. We decide to go one step further and describe an approach in which we decompose system in terms of services, and propose a service choreography paradigm as a way to enable security case run-time adaptation.

B. Security Process

A security case needs to be developed starting from the design phase and therefore we align our mapping approach with the security process. The process presented in Fig. 1 is based on the process introduced by Kizza [17] and adapted to accommodate system decomposition into services. Note, the process is applied at the system level and not at the service level. Even though we aim to enable run-time adaptation, Fig. 1 presents mostly the design phase of the security process, when argumentation is developed. We illustrate our approach on a rather small example leaving further details for the future work.

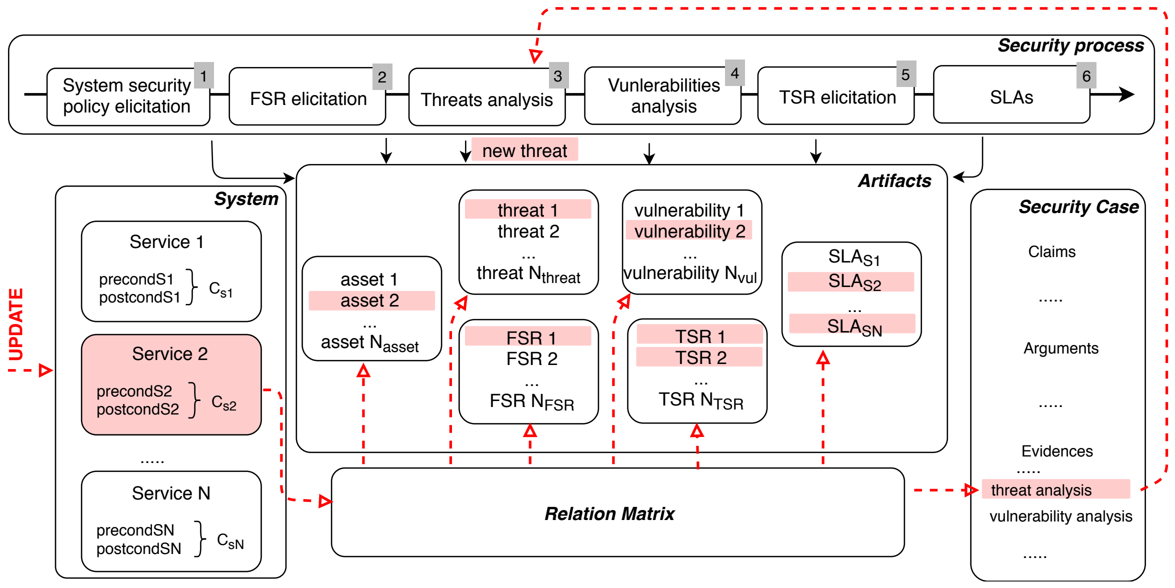

Fig. 1: An example of tracing an update to a security case

The security process consists of consequent steps that may not be strictly continuous, as it allows coming back to previous steps and an additional iterations may be required. In this work we consider security process at run-time as a service choreography. The outcomes of such a security process are called for artifacts. To enable the security process at run-time, beside identified artifacts we need the corresponding security case as well.

Fig. 1 depicts the system decomposed into a set of services. The set is used to map requirements elicited during security process with corresponding services. The security process consists of the following steps: 1) *a system security policy elicitation*, where a security policy is a document incorporating different aspects of system operation, e.g., resources to be protected, rights to access them, actions in case of violations, etc. Along with the process of security policies elicitation, system assets from a security point of view are identified; 2) *elicitation of Functional Security Requirements (FSRs)* to reflex previously formulated policies. For the E-gas system the source of any change in its setting shall be trustworthy, thus a possible requirement can be formulated as following, *FSR1*: *the system shall not provide access to unauthorized users*; 3) *a threat analysis* - a system is analyzed for possible threats and the analysis can be built upon system assets and an adversary model that needs to be formulated to reason about possible risks, resulting in a set of possible threats; 4) *a vulnerability analysis* of the system, that also includes vulnerabilities assessment, i.e., risk estimation of vulnerabilities exposure needs to be performed and a set of system vulnerabilities derived; 5) *Technical Security Requirements (TSRs) elicitation* needs to be performed based on the previously conducted analyses, resulting in a set of TSRs mapped with corresponding FSRs. For example, *FSR1* can be further refined into *TSR1: any change in the E-gas configuration shall be logged;* to be able to trace back changes and *TSR2: user credentials needs to be verified to allow the access to configuration settings;* to limit a number of users allowed to change any setting in the vehicle operation; 6) *capturing of requirements in corresponding SLAs*. We assume that every system that is a service composition in our case, has established an SLA regarding quality, availability, responsibilities, and security requirements that involves all services in such a system. Each SLA includes a security agreement related to a pre-, and postcondition of the corresponding service.

C. Mapping

Outcomes of different steps of the security process we call artifacts. Possible connections between artifacts itself, as well as between services and artifacts are captured in the relation matrix, that is a set of tables carrying this information about all possible connections between artifacts and between services and artifacts. In case an update is introduced to the system that is described in terms of system composition, we need to determine which services are affected in order to be able to do the proper mapping between the update and required re-examinations. Except a service directly affected, all services belonging to a branch with the affected service as a root are marked as affected. The service connections can be traced through SLAs that enable establishing service compositions via service choreographies. In this case we have to consider also type of the connections between service. In case of the serial composition all services that come after the affected service are marked as affected services as well, which might provide disruptions in system operation. Parallel connection increases the redundancy, and enables system to run using non-affected services in case an affected service is identified in the composition. Next, all affected services are used as an input to the relation matrix that carries information regarding assets, requirements, threats and vulnerabilities associated with the affected services. Information regarding these artifacts is acquired throughout the security process from the beginning. Note, that once an artifact is marked as associated with an affected service, this is propagated to all other artifacts it is connected to. In Fig. 1, the artifacts associated with the service *Service* 2 affected by an update, are marked with the red color.

The artifacts derived during the security process are linked to security case parts, e.g., goals can be connected with systems assets, arguments and evidences to the conducted analyses and thus to sets of threats and vulnerabilities. Once artifacts affected by the update are identified they are traced to the security case, identifying affected parts that need to be re-examined. As we only sketch the approach we do not go into details of a re-examination. However, it is worth of saying that a re-examination of a part of the security case may require redoing a whole step in the security process in which this artifact has been derived, or even consequent steps. In Fig. 1, it is the threat analysis that needs to be revised. As an outcome, a new set of threats is derived, which requires a relation matrix extension and, thus a new iteration of the analysis to consider connections between the identified threat and other artifacts and services.

Actual implementation of mapping is a challenging task as we foresee challenges of dependences tracing due to systems complexity. Thus, proper structure and use of the relation matrix is crucial. Derivation of the relation matrix is a meticulous task and has to be supported by a rigorous process. Hence, to enable this process, security process should be in place and aligned with the system development process and its lifecycle.

V. An example — E-gas

In this section we introduce an example of an E-gas system [40] that stands for Electronic Glow Adjustable Switch. E-gas is replacing a mechanical accelerator cable used to connect the accelerator pedal and the throttle in a vehicle. Based on the accelerator pedal, E-gas detects a drivers intention and informs engine control unit that adjusts the opening and closing of throttle.

We decompose an E-gas system into three services corresponding to inputService (iS) that is an accelerator pedal, logicService (lS) implemented through the engine control unit, and outputService (oS) in the form of an e-throttle, as depicted in Fig. 2. Each of these services has a well defined service interface that contains information regarding service functionality and extra-functional properties exposed via a *service precondition* that constrains the service execution (e.g., to calculate the vehicle target acceleration, the power train transmission ratio and vehicle parameters such as vehicle reference mass, reference air drag coefficient, etc. shall be known), and a *service postcondition* that is an output guarantee and has to hold after the execution of a service (e.g., the vehicle acceleration is below the setpoint vehicle acceleration). For the correct functioning of a composed service depicted in Fig. 2, postcondition of inputService (iS) should imply the precondition of logicService (lS) and so on, for the whole composition. Note, that different types of compositions (i.e., parallel composition) have different implication rules, when it comes to handling pre-, and postcondition information. Given that a change in the system occurs (i.e., a security patch or an update installed), and that we are able to identify where the change has occurred (i.e., which services are affected), we should be able to reason about whether such change is strengthening or weakening an already existing assurance case, as well as what is the impact of such change to other services in the systems (i.e., whether the system functions as expected).

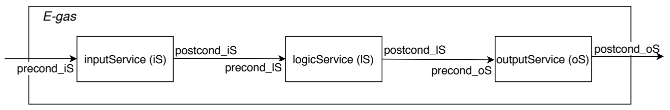

Fig. 2: E-gas system service decomposition

Given that for the maintenance purposes E-gas might be connected to the Internet, an opportunity for a malicious adversary to tamper security in the system exists, i.e., malicious vehicle acceleration or loss of braking control due to adversary activities. In case this would happen at run-time, *logicService* would be affected. To avoid this, we need to trace how an update that prevents such an activity, is related to services, to which artifacts and, consequently, to which parts of the security case those services are connected to and thus, whether the security process described in IV-B needs to be re-examined and to which extent.

The main goal of this process would be to protect control parameters from being changed to unexpected values, and therefore we could define the following *FSR*: *unauthorized users should not have granted access to logicService*. At the same time we could engage threat and vulnerability analyses to investigate what are the other services in the system that might be affected by unauthorized access, assuming that the system enters the vulnerable state. Based on the findings in the analysis and already established FSR, we can elicit the following *TSR1*: *only user with credentials belonging to a specific security level can access and change the configuration of logicService for maintenance purposes*, and *TSR2*: *all changes in the configuration of logicService have to be recorded into a file, to enable future change backtracking*. Both functional and technical requirements have to be reflected in the SLA that is connected to *logicService* such that the initial guarantees on the service qualities are checked and preserved.

In case an adversary would compromise *logicService* by affecting the vehicle acceleration or setpoint vehicle acceleration, we would need to trace effects of such an impact in the whole system using the process described in Fig. 1. In the example provided in Fig. 2, one can notice that *outputService* is directly compromised by such a change and therefore all artifacts connected to this services, as well as the security case, will need a re-examination. Based on this, we can notice that in the case of *logicService* being compromised by a new threat, the threat analysis has to be revised, providing us with a new threat to be added to the list, as well as the relation matrix being extended. In this way, the newly discovered threat will be considered with respect to other artifacts and services in the system.

Assuming that the whole system is decomposed in a similar manner as described for E-gas, and provided that for each service, service interfaces are well defined, carrying correct and accurate information regarding assumptions and output guarantees, one can notice the benefits such an approach brings. We would be not only able to decompose the system in order to identify system parts affected by the change, but also in position to provide run-time assessment of confidence in newly discovered finding in the system with respect to the security.

VI. Conclusions

An acceptable level of system security needs to be guaranteed for safety-critical systems. To argue over system security, a security case can be developed as a collection of arguments supporting system high-level security goals. However, frequent updates, e.g. required due to a newly discovered vulnerability or attack, indicate a need for security case run-time adaptation. As it can be concluded based on the current state of the art in security assurance, the first step towards enabling such an adaptation is the development of an approach for mapping an update to affected service by identifying affected parts of the security case. This paper proposes to facilitate this mapping by system decomposition into services. Building upon a system representation via a choreography of services we can trace an update through artifacts derived from the security process to the security case and identify parts that need to be re-examined.

Further, we plan to detail the proposed approach of mapping an update to a security case through services compositions and artifacts by developing an argument pattern that takes into account services decomposition. Thus, the second identified sub-challenge related to security case structure needs to be addressed as the next step. We have also noticed that in order to enable security assurance and building assurance case in such a way, we need to extend the re-examine the current service interface and enable the notion of security relevant attributes.

Acknowledgements

This work is supported by the SeCRA project funded by Vinnova, the SAFSEC-CPS project funded by KKS and the Serendipity project funded by SSF.

References

[1] C. B. Weinstock, H. F. Lipson, and J. Goodenough, “Arguing security creating security assurance cases,” 2014.

[2] T. P. Kelly, “Arguing safety – a systematic approach to managing safety cases,” 1998.

[3] N. R. Storey, *Safety Critical Computer Systems*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1996.

[4] P. Johnson, D. Gorton, R. Lagerström, and M. Ekstedt, “Time between vulnerability disclosures: A measure of software product vulnerability,” *Computers & Security*, 2016.

[5] E. Denney, G. Pai, and I. Habli, “Dynamic safety cases for through-life safety assurance,” in *37th IEEE International Conference on Software Engineering*, 2015.

[6] D. Schneider, E. Armengaud, and E. Schoitsch, “Towards trust assurance and certification in cyber-physical systems,” in *Computer Safety, Reliability, and Security*, 2014.

[7] B. H. C. Cheng, K. I. Eder, M. Gogolla, L. Grunske, M. Litoiu, H. A. Müller, P. Pelliccione, A. Perini, N. A. Qureshi, B. Rumpe, D. Schneider, F. Trollmann, and N. M. Villegas, *Using Models at Runtime to Address Assurance for Self-Adaptive Systems*. Springer International Publishing, 2014.

[8] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann, “Service-oriented computing: State of the art and research challenges,” *Computer*, 2007.

[9] North Atlantic Treaty Organization, “Engineering for system assurance nato in programmes,” 2010.

[10] R. Weaver, J. Fenn, and T. Kelly, “A pragmatic approach to reasoning about the assurance of safety arguments,” in *8th Australian Workshop on Safety Critical Systems and Software*, 2003.

[11] I. Šljivo, E. Lisova, and S. Afshar, “Agent-centred approach for assuring ethics in dependable service systems,” in *13th IEEE World Congress on Services*, Jun. 2017.

[12] R. Gallo and R. Dahab, “Assurance cases as a didactic tool forinformation security,” in *Information Security Education Across the Curriculum*, 2015.

[13] T. Kelly and R. Weaver, “The goal structuring notation–a safety argument notation,” in *Dependable Systems and Networks*, 2004.

[14] Adelard, “ ASCAD: The Adelard Safety Case Development Manual,” 1998.

[15] ISO/IEC JTC1/SC27, “Common Criteria for Information Technology Security Evaluation, ISO/IEC 15408,” 2005.

[16] M. Broy, I. H. Krüger, and M. Meisinger, “A formal model of services,” *ACM Trans. Softw. Eng. Methodol.*, Feb. 2007.

[17] J. M. Kizza, *Security Assessment, Analysis, and Assurance*. Springer, 2017.

[18] I. Agudo, J. L. Vivas, and J. López, “Security assurance during the software development cycle,” in *Proc. International Conference on Computer Systems and Technologies*, 2009.

[19] M.-Y. Nam, J. Delange, and P. Feiler, “Integrated modeling workflow for security assurance,” in *Leveraging Applications of Formal Methods, Verification and Validation: Foundational Techniques*, Springer, 2016.

[20] K. Taguchi, N. Yoshioka, T. Tobita, and H. Kaneko, “Aligning security requirements and security assurance using the common criteria,” in *4th International Conference on Secure Software Integration and Reliability Improvement*, 2010.

[21] H. Li, X. Li, J. Hao, G. Xu, Z. Feng, and X. Xie, “Fesr: A framework for eliciting security requirements based on integration of common criteria and weakness detection formal model,” in *IEEE International Conference on Software Quality, Reliability and Security*, Jul. 2017.

[22] R. N. Akram, K. Markantonakis, and K. Mayes, “A dynamic and ubiquitous smart card security assurance and validation mechanism,” in *25th International Information Security Conference*, 2010.

[23] A. Hecker and M. Riguidel, “On the operational security assurance evaluation of networked it systems,” in *9th International Conference and Second Conference on Smart Spaces*, 2009.

[24] M. Ouedraogo, H. Mouratidis, E. Dubois, and D. Khadraoui, “Information systems security criticality and assurance evaluation,” in *Advances in Computer Science and Information Technology*, 2010.

[25] M. Ouedraogo, R. M. Savola, H. Mouratidis, D. Preston, D. Khadraoui, and E. Dubois, “Taxonomy of quality metrics for assessing assurance of security correctness,” *Software Quality Journal*, Mar. 2013.

[26] D. Pierce and J. Littlefield-Lawwill, “Composition of information assurance properties in integrated modular avionics systems,” in *27th IEEE/AIAA Digital Avionics Systems Conference*, Oct. 2008.

[27] S. Formoso and M. Felici, *Evidence-Based Security and Privacy Assurance in Cloud Ecosystems*. 2016.

[28] M. Rak, “Security assurance of (multi-)cloud application with security sla composition,” in *Green, Pervasive, and Cloud Computing*, 2017.

[29] I. Rauf and E. Troubitsyna, “Towards a model-driven security assurance of open source components,” in *Software Engineering for Resilient Systems*, 2017.

[30] B. Meyer, “Applying ’design by contract’,” *Computer*, Oct 1992.

[31] M. Montenegro, A. Maña, and H. Koshutanski, “Improving security assurance of services through certificate profiles,” in *Advances in Service-Oriented and Cloud Computing*, 2013.

[32] C. Harris, M. Parsons, and A. Simpson, “Service-based safety assurance,” in *Safety-Critical Systems Club*, 2018.

[33] K. Netkachova, R. Bloomfield, P. Popov, and O. Netkachov, “Using structured assurance case approach to analyse security and reliability of critical infrastructures,” in *Computer Safety, Reliability, and Security*, 2015.

[34] R. Bloomfield, P. Bishop, E. Butler, and K. Netkachova, “Using an assurance case framework to develop security strategy and policies,” in *Computer Safety, Reliability, and Security*, 2017.

[35] T. Saruwatari and S. Yamamoto, “Creation of assurance case using collaboration diagram,” in *Information and Communication Technology*, 2014.

[36] B. D. Rodes, J. C. Knight, and K. S. Wasson, “A security metric based on security arguments,” in *5th International Workshop on Emerging Trends in Software Metrics*, 2014.

[37] P. J. Graydon, “Towards a clearer understanding of context and its role in assurance argument confidence,” in *33rd International Conference on Computer Safety, Reliability, and Security*, 2014.

[38] B. D. Rodes, J. C. Knight, A. Nguyen, J. D. Hiser, M. Co, and J. W. Davidson, “A case study of security case development,” in *Safety-Critical Systems Symposium*, 2015.

[39] A. Nguyen-Tuong, J. D. Hiser, M. Co, N. Kennedy, D. Melski, W. Ella, D. Hyde, J. W. Davidson, and J. C. Knight, “To b or not to b: Blessing os commands with software dna shotgun sequencing,” in *10th European Dependable Computing Conference*, May 2014.

[40] EGAS Workgroup, “Standardized E-Gas Monitoring Concept for Gasoline and Diesel Engine Control Units,” 2013.