Continuous Secure Software Development and Analysis

Sophie Schulz[1], Frederik Reiche[1], Sebastian Hahner[1] and Jonas Schiffl[1]

[1]*KASTEL – Institute of Information Security and Dependability, Karlsruhe Institute of Technology, Karlsruhe, Germany* – `firstname.lastname@kit.edu`

Abstract

Software security becomes increasingly important nowadays. Security should be considered as early as possible in the software development. However, considering different aspects of security is a complex task. In this paper, we propose an extendable framework for continuous secure software development and evolution. The framework provides interconnected analyses on different stages of development. Explicit assumption management helps to verify the security requirements more properly. Thus, the security of the system under development can be estimated more accurately. Finally, the concrete assumptions also help to identify and close security gaps that arise during the software's lifetime.

Keywords

Continuous Security, Security by Design, Access Control, Security Analysis, Palladio Component Model

1. Introduction

In a world of interconnected networks, security becomes increasingly important. Personal data, corporate secrets, or system operations have to be protected from unauthorized access. Thereby, "security should be explicitly at the requirements level" [1]. Appropriately formulated security requirements (i.e., abuse cases) are falsifiable and therefore form the foundation for secure system development. For security, as well as for all requirements, the rule applies: The later a violation is detected, the more expensive the repair gets [2]. Security requirements should be considered and reviewed from the beginning of the software development. However, this is often not the case [3]. If security is not considered in the design stage, then less context is transmitted to the next stages. Security analyses in the implementation stage miss some security aspects and run independently from previous decisions, limitations, and assumptions. However, security is often not treated as an intrinsically dependent construct. The underlying security models often contain only few security aspects [4]. Thereby, different security models and analyses need to be combined to provide an appropriate coverage. This also increases the complexity and effort for the developers to keep track of used mechanisms, covered requirements, and the interrelationships between different security aspects or assumptions. Another problem regarding secure software development is that security is an evolving risk. Thus, assumptions on parts of the system, security mechanisms, and attacker can change during the development or the system's lifetime. Therefore, it is important to check constantly whether assumptions are still valid and whether the system still fulfills its requirements.

SSP'21: Symposium on Software Performance, November 09–10, 2021, Leipzig, Germany

In this paper, we propose a holistic framework for secure software development and evolution. Such frameworks can be classified based on different characteristics: Alkussayer and Allen [5] state that there are four bases to perform security analyses on architectural level: Mathematical modeling, simulation-based, scenario-based, and experience-based analyses. Ryoo et al. [6] state that there are three orientations of architectural security analyses: Vulnerability-oriented, pattern-oriented, and tactic-oriented approaches. Alkussayer and Allen [5] present a scenario-based framework built from different vulnerability-oriented approaches.

Ryoo et al. [6] state that a combination of all orientations should be preferred: At first, they run a vulnerability-oriented approach based on an expert interview. The results are transmitted to a pattern-oriented analysis that analyses design patterns regarding the identified vulnerabilities. Finally, they use a tactic-oriented approach to investigate the handling of attacks.

Khan [7] explicitly checks the security aspects of every development stage before the next stage begins. If the current stage leads to problems, they have to be revised. Khan begins with the requirement phase. It contains a misuse case analysis that is used to verify the previously defined requirements. In the design phase, these misuse cases and a vulnerability analysis are used to perform threat modeling. Regarding the results of this stage (i.e., identified threats) the design can be adapted. The coding phase is closed by a testing phase, including static analyses and code reviews.

However, these frameworks focus on the development, but not the evolution.

The framework we suggest can be applied over time and react to changing requirements or contexts. In contrast to the related work, the interchange of the analyses uses the blackboard principle: All information is managed in a single model, the Palladio Component Model (PCM) [8] which is an established metamodel for software architectures. By extending the model with security information, the exchange of analyses is simplified. At the same time, the complexity for the software architect is lowered. Through the single source, the framework is extendable and has exchangeable analyses. Moreover, analyses can benefit from each other's results.

2. Vision

Our framework begins with the definition of the software system in the form of requirements (c.f. Figure 1) and context information. This contains, e.g., information about data localization, law, and areas of application. Since the architecture is not designed in this stage, this information cannot be linked to specific components. Therefore, given goals and other information are refined to a description as explicitly as possible. In our work, we follow the idea of Broadnax et al. [9]: Refine such goals as much as possible, so that only black box mechanisms and underlying assumptions remain. Thereby, the soft goals of the stakeholders become hard goals that relate to a functional requirement.

During the design phase, an architecture is created and specified as a PCM. By linking these hard goals to components of the architecture, software architects already obtain an overview of the required security. This can reduce complexity and simplify purchasing external components. Based on the architecture, the hard goals of the previous stage can be refined to black box mechanisms. These black boxes (e.g., a VPN protocol) with underlying assumptions (e.g., the correctness of the VPN protocol) can then be linked to specific components. Furthermore, additional assumptions can be annotated based on the work of Taspolatoglu and Heinrich [10]. Based on this extended PCM, multiple analyses can be executed, and the assumptions are falsified. For example, regarding the confidentiality & uncertainty analysis multiple outcomes could occur: On the one hand, the assumption could be verified, because the component has no confidentiality problems. On the other hand, it could also be that this statement depends on another component. Therefore, the existing assumption should be refined to two assumptions (one for the component and one for the dependency). Finally, it could be possible that the given assumption is not met and therefore, the design has to be changed. When all approaches of the design stage have been run, the architecture can be implemented. Thereby, static code analyses or formal verification could check whether the assumptions are met (e.g., by verification whether a security mechanism works correctly). The results of them are transmitted back to the PCM. However, the interrelationship and composition of the architecture and code-based security analyses have to be clarified. The PCM stays consistent as an overview of the current state of the software's security. As mentioned previously, there are some assumptions that cannot be verified automatically (for example the localization of the servers). The PCM would also contain these assumptions which helps in documentation and manual verification.

This principle does not change over time. If, for example, a server location becomes insecure the concerned assumptions could be changed. Since the interdependencies are modeled in the PCM, the impact of the change can be comprehended by executing the framework on the updated model. Accordingly, this works during software evolution and design changes.

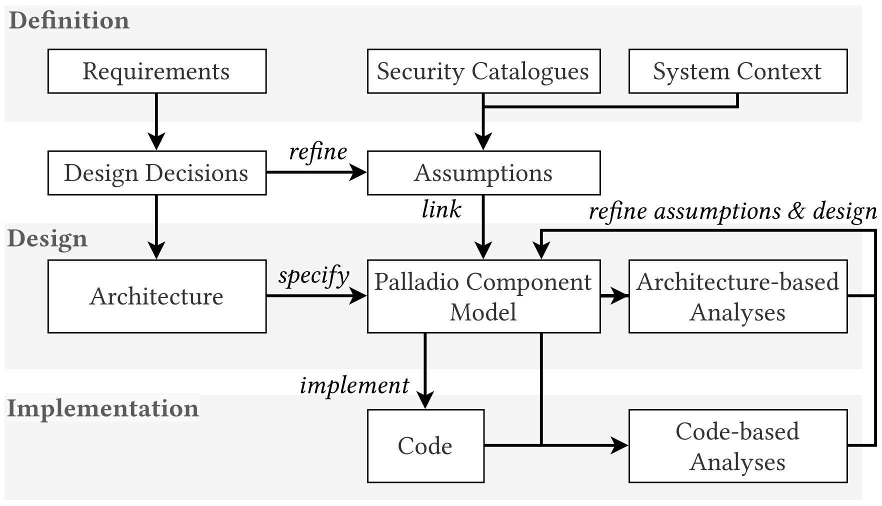

Figure 1: Application of the framework on the software development process

2.1. Approaches

In this section, we propose first approaches for our framework. This includes uncertainty-aware confidentiality analysis, combining code and architecture-based security analyses, and the analysis of assumptions based on formal verification.

2.1.1. Architecture-based Confidentiality Analysis under Uncertainty

Especially in the early phases of the software architecture, design decisions are heavily affected by the subject of uncertainty [11]. While this can have negative impact on many quality attributes like performance and reliability, it becomes increasingly critical regarding security-related properties like confidentiality. Confidentiality demands that "information is not made available or disclosed to unauthorized individuals" [12] and is often ensured by access control.

The problem of uncertainty in software design has already been discussed in a multitude of works, especially in the domain of self-adaptive systems [13]. Uncertainty-aware approaches have been proposed on architectural abstraction [14] as well as in the context of access control [15]. However, these approaches lack the continuity from the requirements phase to the implementation and existing approaches to deal with the requirements-based refinement of access control policies do not explicitly consider uncertainty [16].

We envision a continuous process from high-level confidentiality requirements to low-level access control policies based on their uncertainty-aware refinement and verification against the modeled software architecture [17]. Based on the Palladio modeling approach [8] and existing taxonomies of uncertainty [18], we plan to identify the impact of different sources of uncertainty on the system's confidentiality. The resulting policies are verified, e.g., by formulating them as constraints for data flow analyses [19] or by leveraging code-based analyses.

2.1.2. Composition of Architecture and Source Code View Static Security Analyses

Most security modeling and analyses approaches use models of the system consisting of a certain view, e.g., architecture or source code, as well as security-relevant information. However, most of these models contain information of a single or small number of security aspects [4]. We argue that one reason for such limitation is, that security aspects are better analyzable on a certain view. For instance, the application of secure protocols can be declared in the architectural model, but their correctness and security can only be analyzed by model checkers or other methods [20]. Thus, an architectural analysis that takes the application of protocols into account has to assume their security. Because of the focus on a small number of security aspects in a analysis, the results have to be calculated based on assumptions about other aspects that are not analysed (e.g., because they cannot be represented on the applied view). When these assumptions do not hold, the result of the analysis may contain false results.

We research how the combination (hereafter called *composition*) of static security analyses of the system views architecture and source code can be used to reduce failures arising through assumptions of analyzable aspects. The need for composition of analyses and verification techniques is already stated by Dwyer and Elbaum [21]. Through composition we expect to achieve more comprehensive results by evaluating the assumed aspects on the architectural level through the utilization of source code analyses results. To achieve a composition, the transfer of information between two analyses has to be enabled. For this purpose, we propose a blackbox approach, where only the models and the results are used for the connection and thus, no modification of the underlying analyses have to be performed. Furthermore, we research how it can be determined if the composition of two or more analyses is valid.

2.1.3. Source-level Specification and Verification of Security

Which security properties must be specified and/or verified on the source code level? To answer this question, we aim to identify patterns for security specification and verification. This encompasses recurring formal specification of security properties and building blocks of code that have been proven to fulfil a given security property. While some properties depend on the concrete programming language, others are language-independent, or they can be formalized on a language-independent level and instantiated for concrete applications.

On the architecture level, security properties are assumed or required, and linked to specific components. In order to justify an assumption or show that a security requirement is met, it has to be proven that a component works as expected. Here, formal methods can be used: If a security requirement of a component can be translated into a formal specification of its source code, theorem provers can show that the code fulfills its specification, and thereby that the component fulfills its security requirement. Furthermore, protocol verification mechanisms can prove that a protocol—within or between components—fulfills given security properties.

One concrete recurring security pattern is access control. In a first step, we analyzed how access control policies for Solidity smart contracts can be modelled on an architecture level, and how adherence to the policy can be forced through generation of code and formal specification [22]. We aim to generalize our existing approach to other applications and programming languages.

3. Conclusion

In this paper, we suggest a holistic framework for secure software development and evolution. In the definition phase, requirements and information about contexts or security mechanisms are specified as assumptions. In the design phase, the architecture is specified as PCM, and the assumptions are linked to related components. Based on the PCM, security analyses can run, use the assumptions to evaluate their aspects more specifically, and refine the assumptions they relate to. Accordingly, in the implementation phase, code-based analyses can run and update the information in the PCM. Thereby, software architects can get feedback about the security of the design as early as possible, as well as an overview of the underlying assumptions. This shall reduce complexity and provide an explicit security management.

Acknowledgments

This work was supported by funding from the topic Engineering Secure Systems of the Helmholtz Association (HGF) and by KASTEL Security Research Labs (46.23.01 Methods for Engineering Secure Systems).

References

[1] G. McGraw, Software security, IEEE Security & Privacy 2 (2004) 80–83.

[2] B. Boehm, V. Basili, Software defect reduction top 10 list, Computer 34 (2001) 135–137.

[3] R. Cope, Strong security starts with software development, Network Security (2020) 6–9.

[4] A. van den Berghe, et al., Design notations for secure software: a systematic literature review, Software & Systems Modeling 16 (2017).

[5] A. Alkussayer, W. H. Allen, A scenario-based framework for the security evaluation of software architecture, in: ICCSIT, volume 5, 2010, pp. 687–695.

[6] J. Ryoo, et al., Architectural analysis for security, IEEE Security & Privacy 13 (2015) 52–59.

[7] R. Khan, Secure software development: a prescriptive framework, Computer Fraud & Security 2011 (2011) 12–20.

[8] R. H. Reussner, et al., Modeling and Simulating Software Architectures: The Palladio Approach, The MIT Press, 2016.

[9] B. Broadnax, et al., Eliciting and refining requirements for comprehensible security, in: 11th Security Research Conference, Fraunhofer Verlag, Berlin, 2016, pp. 323–330.

[10] E. Taspolatoglu, R. Heinrich, Context-based architectural security analysis, in: WICSA, 2016, pp. 281–282.

[11] S. Hahner, Dealing with uncertainty in architectural confidentiality analysis, in: Proceedings of the Software Engineering 2021 Satellite Events, 2021, p. 1–6.

[12] ISO, ISO/IEC 27000:2018(E) Information technology – Security techniques – Information security management systems – Overview and vocabulary, Standard, 2018.

[13] S. Mahdavi-Hezavehi, et al., Uncertainty in self-adaptive systems: A research community perspective, ACM Transactions on Adaptive and Autonomous Systems (2021). To appear.

[14] N. Esfahani, et al., GuideArch: Guiding the exploration of architectural solution space under uncertainty, in: ICSE, 2013, pp. 43–52.

[15] T. Bures, et al., Capturing Dynamicity and Uncertainty in Security and Trust via Situational Patterns, in: ISoLA, 2020, pp. 295–310.

[16] Q. He, A. I. Antón, Requirements-based Access Control Analysis and Policy Specification (ReCAPS), Information and Software Technology 51 (2009) 993–1009.

[17] S. Hahner, Architectural access control policy refinement and verification under uncertainty, in: ECSA, 2021. Accepted, to appear.

[18] D. Perez-Palacin, R. Mirandola, Uncertainties in the modeling of self-adaptive systems, in: ICPE, 2014, pp. 3–14.

[19] S. Hahner, et al., Modeling data flow constraints for design-time confidentiality analyses, in: ICSA-C, 2021, p. 15–21.

[20] C. Hochreiner, et al., Genie in a model? why model driven security will not secure your web application., JoWUA 5 (2014) 44–62.

[21] M. B. Dwyer, S. Elbaum, Unifying verification and validation techniques: Relating behavior and properties through partial evidence, in: FoSER, 2010, p. 93–98.

[22] F. Reiche, et al., Modeling and Verifying Access Control for Ethereum Smart Contracts, Technical Report, Karlsruher Institut für Technologie (KIT), 2021.