Poster: Specification and Verification of Confidentiality in Component-Based Systems

Max E. Kramer, Anton Hergenröder, Martin Hecker, Simon Greiner, Kaibin Bao
Karlsruhe Institute of Technology, Germany
{firstname.lastname}@kit.edu

I. Introduction and Motivation

In distributed software systems, confidential data that passes limits of logical components, physical machines, and communication stacks can be a major security problem. To avoid unintended information leaks, the flow of confidential data has to be considered during system design. Current approaches either analyze inter-component or intra-component data-flows and focus on specific concerns, such as protocols [1], policies, or generated code. Whether a component-based distributed system fulfills a confidentiality specification depends, however, also on the *combination* of inter- and intra-component data flow. Therefore, an integration of sound analysis methods has to examine components, their interconnection, and communication. We propose an approach for the combined design and analysis of component-based distributed systems, which integrates architectural confidentiality specifications, adversary modeling, static code analysis techniques, and formal code verification. Our approach enables early confidentiality analyses for complete system architectures with respect to adversaries with concrete capabilities. It allows gradual abstraction from particular functionality, hardware, and communication protocols. The proposed analyses check a) whether the system architecture yields access to confidential data for adversaries, b) whether components refine the specification of composite components and systems, and c) whether confidential information does not interfere with public information in the final implementation. These analyses reveal confidentiality violations in the design and realization of component-based distributed systems.

In the following sections we first explain how confidentiality can be specified for systems in terms of information flow specifications for components and abstract properties for hardware and connectivity. Then, we propose adversary modeling aligned to these system properties. Afterwards, we explain how confidentiality violations can be found with architectural analyses and information flow code verifications.

II. Architectural Confidentiality Specification

In order to enable confidentiality analyses of the complete system, the confidentiality requirements are specified in architectural models. These models represent software components, hardware devices, and communication infrastructure using general concepts of the Palladio Component Model [2], such as software components, hardware resource containers, and links.

This work was funded in the KASTEL project by the German Federal Ministry of Education and Research under grant BMBF 01BY1172.

A. Software Systems and Components

Confidentiality requirements for a system are defined for parameters and return values of services provided by the system. For each in- and output of a service, it is defined which users may directly or indirectly gain knowledge about the passed value. Let, for example, a smart home system provide the service $\mathrm{switchOn(x)}$ to allow a user to switch on an appliance x. By adding x to the information set $\mathrm{inhabitant}$, a user in the role inhabitant may gain knowledge about the value of x, i.e. the information which device was switched on, whereas other users may not. To enforce confidentiality of information encoded in inputs of services provided by components, *non-interference* properties are specified for parameters and return values of services. All parameters and return values of services can be specified to be an element of an information set. The value of an input that is element of a certain information set may at most influence the outputs which are element of the same information set, independent from how the services of the component are used by its environment. Assume our smart home example provides a further service $\mathrm{getBill()}$ to get information about the last energy bill. The return value of $\mathrm{getBill()}$ shall not interfere with input for $\mathrm{switchOn(x)}$. Therefore, the parameter x is specified to be element of the information set $\mathrm{inhabitant}$, whereas the return value of the service $\mathrm{getBill()}$ is not.

B. Hardware Devices and Communication Links

The hardware architecture of a system is modeled on an abstract level with resource containers, which can be connected with links. A resource container may represent arbitrary hardware, such as servers or embedded devices. Deployment of component instances on resource containers is modeled separately. For every resource container four confidentiality properties can be set. The *sharing* property indicates whether additional software components that were not modeled can be deployed on the same machine (*open-shared*) or not (*controlled-exclusive*). The *locality* property represents the physical location of a resource container using system-specific locality classes. A third property states whether a resource container either has *further physical connections* that were not modeled and that are ready to use (*existing*), or if further connections are possible but need to be physically established (*possible*), or if all connections were modeled (*complete*). A last property indicates whether a resource container is equipped with *tamper protection*. Similarly to the locality property, system-specific classes may be used for this property. These tamper protection classes may represent, for example, deadbolt locks or tamper-evident seals.

Links between resource containers represent communication of information. Their influence on confidential data can be specified with the same properties *locality* and *tamper-protection* as for containers and with two additional properties:

Accessibility classes represent in system-specific categories whether access to a link is restricted, for example, in case of password protected WiFi connections. Additionally, a boolean property for *encryption* indicates whether a communication service offers content confidentiality, i.e. end-to-end payload encryption, which means that protocol control information may still be accessible for adversaries. All these confidentiality specifications for containers and links represent indispensable requirements for every possible implementation.

III. Confidentiality Analyses

The architectural model and confidentiality specification is used for three analyses. In preparation of the analyses the adversaries' capabilities are modeled. The first analysis checks whether the system architecture allows direct access to confidential data for these adversaries. The second analysis checks whether the composition of components leads to information leaks. The third analysis verifies that the implementation of the components respects its non-interference requirements.

A. Adversary Modeling

For adversaries the *mayknow*, *location*, *tamper*, and *accessibility* properties have to be defined. They specify of which information sets an adversary is allowed to gain knowledge, which locations he can access, which tamper protections he can overcome, and which communication link categories he can access. These properties limit the capabilities of the considered adversaries for later analyses and the maximum knowledge adversaries may gain about information in the system.

B. Model Transformation and Accessibility Analysis

For the *architectural* analysis, the confidentiality specification is transformed into additional mock-up components and interfaces in order to explicitly represent access possibilities for adversaries. These interfaces express that an adversary may have access to all data of all components that are deployed on a resource container if a) the container is *open-shared* and the adversary has access to its locality, or b) if further connections exist, or c) if further connections are possible and the adversary has access to the containers locality and can overcome at least one of the specified tamper protections. An adversary may have access to the protocol control information and all data passing a communication link, if the accessibility classes of the link and the adversary overlap. An adversary may access the payload, if additionally the encryption property is set to false. The analysis accumulates which information sets an adversary can access by collecting all information sets of all data passing these mock-up interfaces. It finally reports every information set that is accessible by an adversary but is not in the set of information sets he may know. The analysis requires, however, that the components are implemented correctly according to the confidentiality specification. This can be ensured using refinement analysis and code verification.

C. Architectural Refinement Analysis and Code Verification

The *refinement analysis* ensures that the composition of a component satisfies its non-interference properties. We use models and compositionality results for non-interference from literature (e.g. [3]) and extend them with specific properties for our understanding of components, which have shared states. The analysis reports a specification violation, if at least one composition does not adhere to the non-interference specification of the refined component, i.e. there might be an execution of the services that is not non-interferent, even if the components are implemented according to their specification.

While it is convenient to refine components into compositions up to a certain granularity, at some point the components have to be implemented in actual code. We generate Java code stubs for the services provided by a component including a non-interference specification on code level, which is consistent with the specification on component level in the model. The implementation for the services has to be provided manually.

Our approach allows us to apply different *verification methods* and tools for information flow requirements, such as JOANA [4] or KeY [5], to *verify* that an actual implementation of a component satisfies these requirements. We translate the confidentiality requirements of the components into the specification language appropriate to the chosen verification technique: When using the KeY tool, for example, each method will be annotated using KeY's information flow extensions to the Java Modeling Language. Once the implementations of all modeled components are verified, the results of the refinement analysis also apply to the implementation. Code verification reports a data leak, if at least one component is not implemented according to its specification.

IV. Conclusions and Future Work

We presented an integrated method for the specification and analysis of confidentiality in component-based systems. It requires abstract information set specifications for the in- and outputs of a system and its components together with accessibility specifications for hardware and communication links. Based on this confidentiality specification, we proposed continuous analyses, which can be used to improve the architecture and code of a system in order to eliminate data leaks and specification violations. We are currently working on a prototypical implementation of the presented architectural analyses and we are integrating verification and modelling tools [6]. Future research will focus on more precise compositionality properties for non-interference in components and on case studies for the evaluation of scalability and usability. We also strive for soundness proofs for the refinement analysis, the verification techniques for component implementations and the confidentiality of information in the transformed model.

References

[1] J. Jürjens, *Secure systems development with UML*. Springer-Verlag, Berlin, Germany, 2005.

[2] R. Reussner *et al.*, "The Palladio Component Model," KIT, Fakultät für Informatik, Tech. Rep., 2011.

[3] D. Clark and S. Hunt, "Non-interference for deterministic interactive programs," in *Formal Aspects in Security and Trust*, 2008.

[4] C. Hammer and G. Snelting, "Flow-sensitive, context-sensitive, and object-sensitive information flow control based on program dependence graphs," *International Journal of Information Security*, vol. 8, no. 6, 2009.

[5] C. Scheben and P. H. Schmitt, "Verification of information flow properties of java programs without approximations," in *FoVeOOS*, 2011.

[6] M. E. Kramer *et al.*, "View-centric engineering with synchronized heterogeneous models," in *Proceedings of the 1st VAO Workshop*, ACM, 2013.