Challenges to Trading-Off Performance and Privacy of Component-Based Systems

Stephan Seifermann
FZI Research Center for Information Technology
seifermann@fzi.de

Kateryna Yurchenko, Max E. Kramer
Karlsruhe Institute of Technology (KIT)
{kateryna.yurchenko,max.kramer}@kit.edu

Abstract

Determining privacy properties of software systems is essential for certification in certain domains and gains importance for users of software services. Late discovery of degraded privacy properties during development phases makes fixing issues hard and expensive. Approaches that focus on architectural privacy predictions are rare and often do not integrate well with existing tools for performance predictions so that trade-off analyses are not supported. In this paper, we suggest extending the Palladio Component Model (PCM) by means of modeling privacy requirements to support privacy predictions, code generation, and verification, as well as trade-off decisions. The goal of this integration with PCM is to support the development of quality-aware component-based architectures: Our approach will allow trade-offs for privacy *and* performance properties during the early design *and* will ease the verification of the implementation.

1 Introduction

Recent events such as global surveillance disclosures and data thefts in companies raised the privacy awareness of system vendors and its users. To create privacy aware systems cost-efficiently, supporting tools and methods must consider privacy as early as possible in the software development process.

Many approaches including verification [1] or information flow analyses [5] focus, however, on the implementation activities only. A survey on model-based security [7] presents multiple approaches for preserving privacy but they require a fine-grained design, which is not available during the architectural design.

We argue for lifting privacy modeling and analyses from the implementation to the architectural design to facilitate early, cost-efficient, and structured handling of privacy issues, and to support trade-off analyses between privacy and other quality attributes. We are going to extend the Palladio Component Model (PCM) [2] as it provides extensive performance analyses and extension concepts for further quality predictions [8]. Busch et al. [6] already showed that trade-offs for performance and security are feasible with Palladio using estimated security properties. Our envisioned PCM extension provides means of modeling and analyzing privacy of software architectures. Based on this, we discuss challenges for trade-off analyses.

The remainder of this paper is structured as follows: In Section 2, we present our approach for analyzing privacy consisting of a) data flow modeling as analysis foundation, b) analysis for insecurely assembled architectural components, and c) generation of source code and of verification specifications. The challenges in privacy and performance trade-offs are discussed in Section 3. Section 4 concludes the paper.

2 Privacy Modeling and Analyses

We identified four main influence factors for rating architectural privacy: a) How and where does a system process data? b) What are my security requirements? c) Which capabilities does an adversary have? d) Does my implementation conform to the design?

Abstractions of system behavior (a) enable quality analyses. PCM supports abstractions that satisfy performance analysis needs but lacks support for privacy. With the first part of our extension [9], privacy relevant behavior and data flow can be modeled based on existing behavior specification facilities.

Security requirements (b) provide input for privacy analyses (c). The second extension part are adversary models and architectural confidentiality analyses. The analysis uses information flow into data sets for detecting confidentiality breaches. The allowed flows can serve as component behavior specifications, or can be derived from existing behavior specifications.

The implementation must adhere to the privacy-ensuring architecture (d). The third extension is the generation of code stubs that can be manually completed and be verified against the modeled requirements. The behavior and data set specification serves as generation input. This prohibits architectural drift and degraded privacy.

2.1 Data Flow Modeling

Data flows formalize system behavior in a way that enables privacy analyses. We envision data flows as a lightweight, natural behavior specification focused on privacy analyses. A possible definition of confi-

dentiality is that no information may flow from data considered confidential to data not considered confidential. More strictly speaking, no information of data classified as "high" (confidential) may interfere with information of data classified as "low" (non confidential). This is called non-interference. Modeling behavior by data instead of control flows is novel in PCM and requires new model entities.

Classic data flows consist of data sources, sinks, processings, and actual data flows shown as directed edges. We extend these elements to make them sufficient for behavior specifications: Data does not only have a name but also additional meta-data for describing data properties. We do not model individual but classes of data like domain models such as entity-relationship (ER) models. The concrete meta-data depends on the concrete analysis. For confidentiality, the confidentiality level is considerable meta-data. Behavior modeling only covers effects relevant for analyses instead of implementation instructions such as code: Processing operators meme the effect of real data processing by adjusting meta-data. Storages are new PCM elements that meme data containers such as files. Data sources and sinks can be users and storages. A new behavioral specification called Data Flow Service Effect Specification (DF-SEFF) models data processing inside components. Data exchange is possible via operation signatures but architects can associate data with more than one parameter.

The DF-SEFF supports analyses as shown in Figure 1: Data flows extend the existing Architectural Description Language (ADL) artifact and enable analysts to specify analysis goals such as no processing of personal information in certain geographical locations with parameterizable goal templates. A transformation engine maps the extended architecture and goals to constraints. Constraints can be facts such as the physical location of a server, rules that deduce information such as the physical location of a deployed component, or goals such as confidential information must remain in physical locations. A transformation maps raw results of an off-the-shelf constraint solver to comprehensible results for architects. The architect uses the results to improve the architecture.

2.2 Architectural Analysis

We developed an analysis that verifies whether a component-based architecture is designed in such a way that confidential information can be leaked to modeled adversaries. If this analysis does not report a leak, the implementation has to fulfill the architectural requirements to preserve confidentiality. A discovered potential leak means that unwanted explicit or implicit information flows to an adversary are possible. Such an information flow can be avoided by redesigning the architecture *and* implementing it according to the requirements.

Our architectural analysis is based on confidentiality requirements and information flows defined on data sets. An adversary model defines which adversaries are allowed to obtain information from which data sets. For PCM models, stereotype tags specify how information flows from data sets into parameters and from return parameters into data sets. Other tags specify the location of resource containers and linking resources. A comparison of the location tag and the accessibility of locations by the adversary yields accessible resources. Another tag specifies how resources process information, which may cause implicit information flows. Further tags denote whether a resource is protected against tampering and which adversaries are able to break such protections. Based on these additional specifications, the analysis checks whether the system is vulnerable because adversaries could unintentionally obtain information. This is the case if an adversary has access to a resource that processes information that may be influenced by information of a data set for which no allowed knowledge was specified. For every vulnerability, the analysis generates an explanation of how a given adversary may exploit it. The explanation guides architects or domain experts to model and specification elements that lead to the vulnerability.

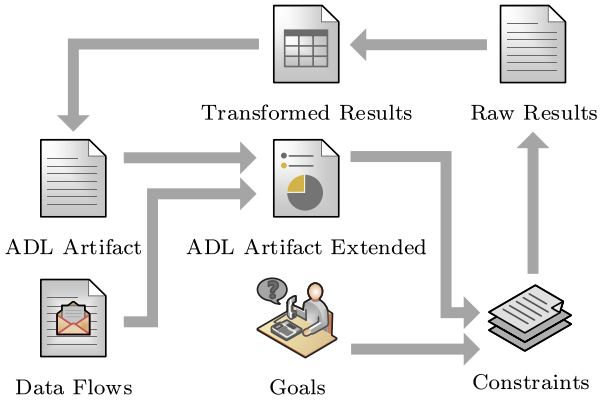

Figure 1: Envisioned data flow analysis approach.

Our architectural confidentiality analysis is implemented using inference rules encoded in Prolog. These inference rules directly implement intuitive requirements for information flows and access to information in the architecture. To use these Prolog rules for PCM models that were extended with our confidentiality profile, we implemented a model-to-text transformation. It generates Prolog facts for relevant elements and properties of these extended PCM models and for adversary models. The code for the analysis and Prolog generator are open-source[1] and our analysis is described in detail in a technical report.

2.3 Implementation Verification

In the specification approach for the analysis described above, an essential step is to add confidentiality requirements to particularize which data sets

[1] sdqweb.ipd.kit.edu/wiki/MD-Confidentiality_in_CBS

shall not interfere. Here, input and return parameters of components must be separated into groups to specify information flow properties implied by them.

The stereotype $\ll d$ *includes* $P \gg$, with data set d and parameters P, annotates services provided or required by components and states that all information of parameters in P is contained in data set d. This yields to the information flow requirement that a data set must not contain information of a parameter, for which no *includes* relation has been defined. Component implementations must obey information separation regarding data sets specified in the architecture.

To verify the implementation with respect to the architecture, we aim to automatically generate source code specifications from the $\ll d$ *includes* $P \gg$ annotations and add them to generated (and eventually manually completed) method stubs. After developers implemented the method stubs, KeY [1] can verify non-interference of the Java program code based on method contracts formulated in a Java Modeling Language (JML) extension.

Software engineers must prohibit architectural drift and degraded privacy during system evolution. Thus, confidentiality requirements that were verified for the original system also have to hold for the modified system. We plan to investigate the applicability of regression verification, in which privacy specifications of the previous program version serve as specifications for the new version. For architectural parts missing security specifications, we envision a method to prove that a new version is at least as secure as the previous one. Hence, developers can verify code modifications using KeY with regard to change specifications.

3 Challenges for Trade-Off Analyses

Trade-off analyses find sweet spots in a bunch of design alternatives that balance relevant quality attributes. Criteria for good balances are highly individual and depend on many factors such as domains, use cases, or external factors. Therefore, optimization approaches such as PerOpteryx [3] often present pareto-sets that contain a small selection of alternatives to the analyst. We identified the following three preliminary challenges for the optimization process itself but would like to discuss further challenges with respect to our modeling and analysis approach:

Design Space Definition specifies available alternatives selectable by the optimization approach. Privacy and performance modeling have to provide design alternatives that specify the effect on privacy *and* performance. In case of privacy, security patterns can serve as design alternatives but have to be augmented with performance effects.

Evaluation Functions allow ordering alternatives with respect to quality. Finding an ordinal scale for security is not trivial in contrast to performance. For privacy, combining exploitation probability with the value of endangered data can be reasonable.

Consistency of Architectural Views enables up-to-date ratings of *all* quality properties after changing an architectural aspect. For instance, changing the location of a node because of data protection laws in the data flow view affects performance properties of the link to this node such as latency and throughput. A consistency preserving approach such as Vitruvius [4] can ease the definition of consistency rules but finding them still remains a challenge.

4 Conclusions

In this paper, we presented our proposed Palladio extensions that allow predicting privacy properties of a system architecture. We exploit the expressiveness and comprehensibility of data flow models to specify component behaviors in a data-oriented, natural way. A confidentiality analysis uses this specification, derives a constraint logic problem, solves it, and thereby ensures confidentiality preservation. Additionally, we presented our vision to enforce the privacy-aware architecture in source code by generating stubs for code verification. We identified design space definition, evaluation function definition, and consistency preservation of architectural views as the major challenges in trading-off performance and privacy.

Acknowledgments

This work was partly supported by the German Federal Ministry of Education and Research within the framework of the project "Security for the Internet of Everything" in the Competence Center for Applied Security Technology (KASTEL).

References

[1] B. Beckert et al., eds. *Verification of Object-Oriented Software: The KeY Approach*. Springer-Verlag, 2007.

[2] S. Becker et al. "The Palladio component model for model-driven performance prediction". In: *Journal of Systems and Software* 82 (2009), pp. 3–22.

[3] A. Koziolek et al. "PerOpteryx: automated application of tactics in multi-objective software architecture optimization". In: *Proceedings of QoSA-ISARCS*. 2011, pp. 33–42.

[4] M. E. Kramer et al. "View-centric engineering with synchronized heterogeneous models". In: *Proceedings of VAO*. 2013, 5:1–5:6.

[5] G. Snelting et al. "Checking Probabilistic Noninterference Using JOANA". In: *it - Information Technology* 56 (2014), pp. 280–287.

[6] A. Busch et al. "Assessing Security to Compare Architecture Alternatives of Component-Based Systems". In: *Proceedings of QRS*. 2015, pp. 99–108.

[7] P. H. Nguyen et al. "An extensive systematic review on the Model-Driven Development of secure systems". In: *Information and Software Techn.* 68 (2015), pp. 62–81.

[8] M. Strittmatter et al. "A Modular Reference Structure for Component-based Architecture Description Languages". In: *Proceedings of ModComp*. 2015, pp. 36–41.

[9] S. Seifermann. "Architectural Data Flow Analysis". In: *Proceedings of WICSA*. 2016, pp. 270–271.