T. Hüller, F. Schwickerath, B.. Arp, N. Niehues, N. Boltz, and S. Hahner, "Towards a Data Flow Diagram-Centric Confidentiality Analysis in Palladio", 16th Symposium on Software Performance (SSP), Softwaretechnik-Trends, Gesellschaft für Informatik e.V., 2024, https://dl.gi.de/handle/20.500.12116/45518

Towards a Data Flow Diagram-Centric Confidentiality Analysis in Palladio

Tom Hüller
tom.hueller@student.kit.edu
Karlsruhe Institute of Technology (KIT)

Felix Schwickerath
felix.schwickerath@student.kit.edu
Karlsruhe Institute of Technology (KIT)

Benjamin Arp
benjamin.arp@student.kit.edu
Karlsruhe Institute of Technology (KIT)

Nils Niehues
nils.niehues@kit.edu
Karlsruhe Institute of Technology (KIT)

Nicolas Boltz
nicolas.boltz@kit.edu
Karlsruhe Institute of Technology (KIT)

Sebastian Hahner
sebastian.hahner@kit.edu
Karlsruhe Institute of Technology (KIT)

Abstract

The Palladio approach enables software architects to create architectural models of their systems for early cost, performance, and maintainability analysis. Using a data flow-based confidentiality analysis, it is also possible to detect confidentiality violations in software systems modeled with the Palladio Component Model (PCM). However, many software architects work directly with Data Flow Diagrams (DFDs) because of their decreased complexity and their ability to make pinpointing specific information security issues easier. To achieve the best of both worlds, a conversion is needed that semantically preserves all security-related information. This paper presents a transformation of PCM instances into information security-annotated DFDs, that can be used by software architects to visualize the data flow analysis results graphically and identify potential confidentiality violations. In our evaluation, we show that the analysis results of the transformed DFDs are equivalent to those of the original PCM instances.

1 Introduction

With an ever-increasing number of data breaches, ensuring data confidentiality has become an integral part of software system development. Data confidentiality must be ensured as early as possible to prevent further upsets in the development process or, even worse, an issue in production software. One possible approach to ensuring confidentiality during the design phase is model-driven software engineering. Models allow software architects to work on a higher abstraction level and give an extensive overview of the system.

One example of model-driven engineering is the Palladio Approach [1]. With Palladio, software architects can create models of their systems for continuous simulation and analysis, enabling them to identify and fix issues early in the development process. With our data flow analysis framework [4], software architects can analyze these models and specify data flow constraints the model has to fulfill to ensure information security.

However, models in the PCM are based on several different metamodels, which complicates pinpointing the reason for an identified violation. A much more straightforward view of the system can be achieved with DFDs. In their work, Schneider et al. [7] show that by employing DFDs, software architects were 41% more likely to identify issues when analyzing a system. An even bigger improvement can be seen in the correctness of evidence for an identified violation.

To allow for a simplified interpretation of the analysis results, we, therefore, propose an approach to transforming PCM instances into information security enriched DFDs that retain all information gained during the analysis and can be used with a provided graphical DFD editor [4] to gain an easy overview of potential violations.

The remainder of this work is structured as follows. Section 2 gives an overview of our approach and we explain how PCM instances are transformed into information security-equivalent DFDs, which may be further employed with our Web Editor. Section 3 covers the validation of our approach, while in Section 4 we draw our conclusion and give an outlook on future work.

2 Converting PCM instances into a Data Flow Diagram

Several steps are needed to transform the PCM instances into DFDs. Figure 1 shows a diagram of the process. The PCM instance is first evaluated with our analysis tool. It creates a graph-based representation of all data flows to their termination point and holds information that is relevant for information security.

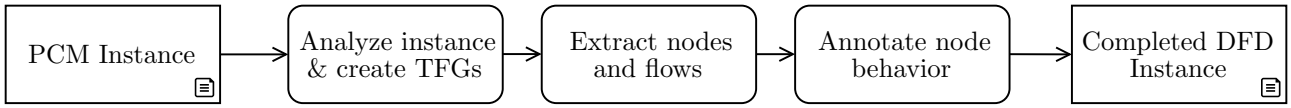

Figure 1: Activity Diagram of Transformation Process

From this graph-based representation, we then extract the DFD nodes and flows and the relevant behavior they display.

2.1 Extended DFD Metamodel

DFDs allow software architects to model all the data flows in their system. They lack, however, certain information-security-related features, like multiple independent flows, that are necessary to visualize confidentiality issues. Seifermann et al. [3] developed their own extended syntax, which has been further enhanced by Boltz, Hahner et al. [4]. In addition to the original elements of nodes and flows, the extended syntax introduces several new elements.

Labels are discrete values that can be either assigned to a node as a property or be passed on a flow as a data characteristic. They represent security-related information like the geographical location of a database or the sensitivity of a specific piece of data and can be propagated through the diagram.

Pins are assigned to nodes and represent input and output interfaces. Each flow needs to start and end in a pin. Multiple flows to the same input pin represent multiple independent routes the data can use to reach the node. These pins, as well as the next new element assignments, are stored in a nodes behavior.

Assignments let the software architect specify which labels are propagated by different nodes. The assignments take the labels that arrive at a defined number of input pins and either simply forward them or adjust them before passing labels on to a specified output pin. This only happens once labels have arrived at all the specified input pins.

2.2 Transpose Flow Graphs

Once the analysis is run on the PCM instance, a collection of Transpose Flow Graphs (TFGs) is returned. The TFGs consist of vertices, representing one point in the system where data is accessed, processed, or handed on. In the flow graph, each vertex contains information about its predecessor vertices. This information represents one route data can take to reach a specified sink vertex. These sinks are points in the system where the flows terminate, so no more data is passed on. This means that if there are multiple ways to reach a specific sink vertex, a TFG is created for each route. Vertices also comprise a collection of vertex characteristics that correspond to the node properties in the DFD syntax, as well as, incoming and outgoing data flow variables. Data flow variables are a collection of data characteristics equivalent to labels in the DFD metamodel.

2.3 PCM to TFG

In the first step of the transformation, the PCM instance is analyzed using our analysis tool, creating the TFGs. Next, the created TFGs are evaluated by propagating the data flow variables through the graph. An in-depth overview of this process can be found in [4].

2.4 TFG to DFD

The actual transformation is handled after the analysis has created and evaluated the TFGs. In the first step, a DFD node is created for each vertex in a TFG. The vertex characteristics are transformed into labels and added as node properties. Then pins are created on each node, one for each incoming and outgoing data flow variable, to facilitate an easy understanding of which route a specific data characteristic might have taken.

The PCM contains control flows that need to be preserved in the DFD for better understanding. To allow this, we assign a dummy pin to the corresponding node of a vertex which is the predecessor of another, but itself does not have outgoing data flow variables.

Next, the flows are created for each data flow variable passed from a previous vertex to its successor. These flows are named after the data flow variables, to allow software architects to quickly connect flow and corresponding data flow variable in the web editor. Again, dummy flows need to be created in case of no flowing variables to preserve the PCM control flows.

In the final step, the node behaviors are inferred from the incoming and outgoing data flow variables on the related vertex. By comparing the incoming to the outgoing variables, we can determine which kind of assignment needs to be created to emulate the PCM behavior. If all incoming variables are returned, no matter their type, a normal forwarding assignment is created. It takes all labels arriving at the specified input pins and returns them through the output pins. PCM additionally supports forwarding only data characteristics of one type, e.g. the sensitivity of data. Since this behavior can not directly be modeled in the DFD, we create an assignment for each individual data characteristic, which returns the corresponding label in case the same label arrives at an input pin. The final kind of assignment we need to create is based on terms. These terms either check for an incoming label or connect other terms through logical operators. Should the term for the assignment be fulfilled, the specified output labels are returned. PCM also employs terms for complex operations from another metamodel, which can be directly transformed

into DFD terms.

To conclude the behavior transformation, we create dummy assignments that make each output pin dependent on every input pin. Otherwise the analysis would detect separate flows and create multiple unconnected TFGs.

2.5 Making the results available in a Web Editor

In a previous work [4], we presented a graphical DFD editor that can visualize a JSON representation of a DFD. The editor allows for annotations and color highlighting of violations and provides a software architect with a simplified way of interacting with the DFD. This work includes a transformation into the JSON representation and back.

3 Validation

To validate the results of our transformation, we employ three different PCM model instances of varying size and complexity. TravelPlanner [2], Mobility-As-A-Service (MaaS) [6] and a model of the open source CoronaWarnApp (CWA) of the German state [5].

Model	#Nodes/ Vertices	#Flows	#Constraints	#Violations
TP	42	80	1	3
MaaS	200	433	5	0
CWA	687	1263	0	0

These models are first put into the analysis and then transformed into DFDs. To compare the results, we first check whether the number of created DFD nodes equals the sum of all vertices in the flow graph collection. We then test whether the labels were created for each data characteristic that was assigned or has been passed on by the vertices. To test for semantic equivalency regarding information security, we run the analysis with the created DFDs and same constraints as for the PCM model and compare the resulting constraint violations. Constraints are predicates that can be handed to a initialized analysis to test every individual vertex for violations of the predicate. The TravelPlanner constraint for example asserts that each vertex has been assigned the appropriate roles for all incoming data.

Our validation results show that the DFDs created by the transformation have the correct number of nodes and labels, and the correct constraint violations are found for all model instances.

4 Conclusion

In this work, we presented our approach on transforming PCM instances into information security-enriched DFDs. In the validation, we showed that the transformation works correctly for the provided test models and that all expected violations, already present on the PCM instances, were found.

Our approach allows software architects to get an analysis of their PCM instance, which can then be viewed as a DFD. As Schneider et al. [7] have shown, software architects work significantly more efficiently with DFDs and can better argue their results. With the help of the graphical DFD editor, the user may even get a visual representation of the DFD, which facilitates quick location of potential information security violations.

In future work, we plan to extend this approach to facilitate a full round-trip transformation. Software architects shall be able to analyze their PCM instance, transform it into a DFD, make the necessary changes to facilitate confidentiality in our graphical editor, and then transform it back into a PCM instance. With this, software architects could design their system in PCM, but then do all the information security relevant work with DFDs before converting any changes back to their PCM model.

Acknowledgements

This publication is partially based on the research project SofDCar (19S21002), which is funded by the German Federal Ministry for Economic Affairs and Climate Action. This work was also supported by funding from the topic Engineering Secure Systems of the Helmholtz Association (HGF) and by KASTEL Security Research Labs, the BMBF (German Federal Ministry of Education and Research) grant number 16KISA086 (ANYMOS), and the NextGenerationEU project by the European Union (EU).

References

[1] R. H. Reussner et al., eds. *Modeling and Simulating Software Architectures – The Palladio Approach*. MIT Press, 2016. 408 pp.

[2] K. Katkalov. “Ein modellgetriebener Ansatz zur Entwicklung informationsflusssicherer Systeme”. doctoralthesis. Universität Augsburg, 2017.

[3] S. Seifermann et al. “Detecting violations of access control and information flow policies in data flow diagrams”. In: *JSS* 184 (Feb. 2022).

[4] N. Boltz et al. “An Extensible Framework for Architecture-Based Data Flow Analysis for Information Security”. In: *ECSA*. Springer. 2023.

[5] S. Hahner et al. “Architecture-Based Uncertainty Impact Analysis to Ensure Confidentiality”. In: *SEAMS*. 2023.

[6] M. Leinweber et al. “Leveraging Distributed Ledger Technology for Decentralized Mobility-as-a-Service Ticket Systems”. In: *TNNM*. May 2023.

[7] S. Schneider et al. *How Dataflow Diagrams Impact Software Security Analysis: an Empirical Experiment*. IEEE Computer Society, Mar. 2024.