A Unified Model to Detect Information Flow and Access Control Violations in Software Architectures

Stephan Seifermann, Robert Heinrich, Dominik Werle and Ralf Reussner
KASTEL – Institute of Information Security and Dependability, Karlsruhe Institute of Technology, Germany

Keywords: Access Control, Information Flow, Software Architecture, Confidentiality, Analysis Automation.

Abstract: Software architectures allow identifying confidentiality issues early and in a cost-efficient way. Information Flow (IF) and Access Control (AC) are established confidentiality mechanisms, so modeling and analysis approaches should support them. Because confidentiality issues often trace back to data usage, data-oriented approaches are promising. However, we could not identify a data-oriented approach handling both, IF and AC. Therefore, we present a unified data-oriented modeling and analysis approach supporting both, IF and AC, within the same model in this paper. We demonstrate the integration into an existing architectural description language and evaluate the resulting expressiveness and accuracy by a case study considering 22 cases.

1 INTRODUCTION

Confidentiality according to ISO 27000 (International Organization for Standardization, 2018) is the "property that information is not made available or disclosed to unauthorized individuals, entities, or processes". Considering security, which includes confidentiality, is vital for software systems (Venson et al., 2019) but is also a challenging because of increasing complexity and connectedness of systems (McGraw, 2006, chap. 1). Addressing issues in early development phases such as the architectural design is more cost-efficient than in later phases such as the implementation in general (Boehm and Basili, 2001; Shull et al., 2002). The same holds for security issues (Microsoft Corporation et al., 2009) in the design (Hoo et al., 2001), (McGraw, 2006, chap. 5).

Automated modeling and analysis approaches can support people reviewing security aspects and identifying such issues with less effort (Tuma et al., 2020). However, this is only true if an approach supports the used security mechanisms and policies. Two established confidentiality mechanisms are Access Control (AC) (Sandhu et al., 1994), (Furnell, 2008, chap. 5) and Information Flow (IF) control (Smith, 2007; Hedin et al., 2012). Both can be used separately or in combination (Xu et al., 2006; Wang et al., 2009). If common and specific parts are distinguished, the advantages of a modeling language supporting both, IF and AC, in one single language are: i) The common part, which could be the structure of the system, only has to be modeled once. Therefore, architects do not have to remodel large parts of the system when deciding for another confidentiality mechanisms. ii) Architects only have to learn the core language and the language elements specific for the chosen confidentiality mechanism. Consistency management between dedicated models might enable some of these benefits as well but it is challenging if languages diverge too much (Torres et al., 2020). Using a data-oriented language in contrast to a control flow oriented language can be beneficial as well because Data Flow Diagrams (DFDs) are commonly used in threat modeling and security "problems tend to follow the data flow, not the control flow" (Shostack, 2014, p. 44).

Surveys (Nguyen et al., 2015; van den Berghe et al., 2017) show a wide range of design-time confidentiality analyses. However, we could not identify a data-oriented approach capable of representing and analyzing IF and AC within the same modeling language and providing the aforementioned advantages. Generic frameworks such as Unified Architecture Framework (UAF) (OMG, 2020) are flexible enough to express such aspects but are too generic to be useful. We see two research questions: RQ1) What modeling primitives can be shared between IF and AC and which primitives have to be specific? RQ2) How can analyses exploit the shared language parts for identifying violations of IF and AC policies?

In this paper, we address both research questions by two contributions: C1) We define the model elements for representing system aspects relevant for

Seifermann, S., Heinrich, R., Werle, D. and Reussner, R.
A Unified Model to Detect Information Flow and Access Control Violations in Software Architectures.
DOI: 10.5220/0010515300260037
In *Proceedings of the 18th International Conference on Security and Cryptography (SECRYPT 2021)*, pages 26-37
ISBN: 978-989-758-524-1

IF and AC. We also discuss how existing Architectural Description Languages (ADLs) based on control flows or data flows can be extended by example. C2) We provide analysis definitions for identifying IF and AC policy violations in such extended ADLs. We realize our concepts within the ADL Palladio (Reussner et al., 2016) in order to identify the required steps to extend existing ADLs. We chose Palladio for two reasons: First, Palladio is capable of describing behavior in terms of control flow and data flow (Werle et al., 2020), so we can evaluate our concepts for both paradigms within one prototype. Second, Palladio provides a condensed set of modeling primitives tailored for analyzing architectural quality properties, which allows us to focus on core concepts.

We evaluate the expressiveness of the modeling primitives and the accuracy of the analysis definitions in a case study. The case study involves modeling 8 systems in 22 cases that include 4 types of IF policies, 4 types of AC models and 1 combined policy. For every case, there is an analysis definition that we execute on a variant of the case featuring a policy violation and one variant free of policy violations to determine the accuracy of our analyses. We could express 20 of 22 cases and correctly identified all injected issues. This means our proposed modeling primitives and the defined analyses are capable of expressing IF and AC in systems and successfully detected all violations.

The remainder of this paper is structured as follows: Section 2 covers foundational knowledge and Section 3 presents the running example. State of the art is covered in Section 4. In Section 5, we determine modeling primitives required for representing IF and AC by comparing two existing approaches supporting either IF or AC. In addition, we discuss how existing ADLs can be extended. Section 6 covers analysis definitions for common IF and AC policies based on the modeling primitives. The expressiveness of the modeling primitives as well as the accuracy of defined analyses are evaluated in Section 7. We also discuss limitations, the implementation and data availability there. Section 8 concludes the paper.

2 FOUNDATIONS

This paper is about an approach for analyzing IF and AC on Data Flow Diagrams (DFDs) and Palladio.

Access Control (AC) (Sandhu et al., 1994) limits actions of users or processes to avoid security breaches. AC mechanisms enforce AC policies describing permissions. Four established AC models (Furnell, 2008, chap. 5) describing policies are Discretionary Access Control (DAC), Mandatory Access Control (MAC), Role-based Access Control (RBAC) and Attribute-based Access Control (ABAC), which we describe later as part of our analysis definitions.

Information Flow (IF) (Smith, 2007) controls the release and the propagation of information. The most prominent IF property is non-interference, which only allows information flows upwards a lattice of security labels, i.e. information on a high level must not influence information on a lower level. This definition is too restrictive for real software systems (Zdancewic, 2004), so mechanisms like declassification explicitly grant flows downwards. Non-interference helps to establish confidentiality and integrity. However, only confidentiality is in the scope of this paper.

DFDs (DeMarco, 1979) describe functional system aspects. DFDs consist of data flows between the following nodes: External entities (sources/sinks) exchange data with the system. The system consists of processes and stores. Processes transform incoming data to outgoing data. Stores save incoming data and emit saved data. Throughout this paper, we assume that processes need all inputs to produce all outputs.

Palladio (Reussner et al., 2016) is a component-based ADL. The language covers the usage, structure, behavior and allocation of a software system. Usage and behavior descriptions are ordered lists of actions that have effects, e.g. on resource usage. The structure consists of components that provide and require services and connected instances of these components. Allocations assign component instances to resource nodes. The Palladio extension for data-oriented modeling (Werle et al., 2020) introduces data channels as special components that communicate by emitting and consuming data items instead of doing calls.

3 RUNNING EXAMPLE

We use the TravelPlanner system (Katkalov et al., 2013; Seifermann et al., 2019) to explain our approach. The system lets users book flights as shown in Figure 1. First, users request flights from their *TravelPlanner* app by giving search criteria. The app passes the request to a *TravelAgency* service that sends a query to the *Airline*. The airline returns matching flights. Users select a flight and retrieve their credit card details *ccd*. The airline is not allowed to access this data, so users declassify it first. Afterwards, users book the flight by submitting the flight and the credit card details. The airline pays a commission to the travel agency and confirms the booking. Simply said, the confidentiality policy is that only users access credit card details except they explicitly grant access. Therefore, the flight booking call from

the travel planner app to the airline is critical.

Katkalov et al. (Katkalov et al., 2013) defined an IF policy by three levels. Level 1 is accessible to the user, airline and travel agency. All data except credit card details are classified by level 1. Level 2 is accessible to the user and the airline. Level 3 is accessible to the user only. Credit card data is level 3. Releasing credit card details means changing its level from 3 to 2. A violation occurs if a system part with clearance level n receives data classified with level $m > n$.

In previous work (Seifermann et al., 2019), we defined a RBAC policy. Roles match the involved parties user, travel agency and airline. Transmitted data holds a set of roles allowed to access. Credit card details only hold the user role. Remaining data is accessible by all. Releasing credit card details means adding the airline role to the data. Violations occur if the system part and data do not share any role.

4 STATE OF THE ART

We focus on providing a unified modeling and analysis approach for detecting IF and AC violations in software architectures. Therefore, we discuss approaches considering IF or AC in architecture or design. We see three groups of related approaches.

Threat Modeling: is an established method for identifying security issues in early software designs. Various models and procedures help designers in identifying potential security issues (Shostack, 2014). Many approaches extend DFDs to increase their expressiveness (Sion et al., 2020) and support automated analyses. A considerable share of these approaches (Abi-Antoun et al., 2007; Berger et al., 2016; Sion et al., 2018) restricts itself to pattern matching that requires manual classification of exchanged data with attributes such as *contains personal data*. In contrast, we focus on deriving this classification automatically to detect confidentiality issues. We put threat modeling approaches with automated classifications into the following last category.

Control Flow Analyses: provide powerful means for identifying confidentiality issues. Almorsy et al. (Almorsy et al., 2013) calculate security metrics to identify structural confidentiality issues. However, they only mention system behavior vaguely, so we assume they do not consider the effect of data processing. There are also approaches (Abdellatif et al., 2011; Katkalov et al., 2013) complementing design information with source code to consider detailed data processing. Besides the increased specification effort, this information might not be available during design time. Additionally, source code is not necessary to consider data processing: Hoisl et al. (Hoisl et al., 2014) define IF analyses comparable with taint analyses. Such analyses usually overestimate information flows, which might yield more false positives than detailed analyses. Approaches like UMLSec (Jürjens, 2005) or the approach of Heyman et al. (Heyman et al., 2012) use more detailed behavior descriptions to detect violations, which allows more fine-grained analyses. However, control flow approaches cannot serve as unified models that also include data flow architectures because mapping data flows onto control flows is not trivial (Alabiso, 1988; Jilani et al., 2011).

Data Flow Analyses: support various types of confidentiality analyses. Data flow models can serve as unified models because mapping control flows on data flows is well known from compiler optimization (Lowry and Medlock, 1969) and there are established mappings (Khedker et al., 2009). Recently, several data flow analyses have been proposed. SecDFD (Tuma et al., 2019; Tuma et al., 2020) uses extended DFDs to detect violations of IF policies as well as other security properties. Van den Berghe et al. (van den Berghe et al., 2017) also detect IF violations but use a Domain-specific Language (DSL) to specify systems and their behavior formally by DFD-like concepts. Both approaches could support the IF analysis of the running example. In previous work (Seifermann et al., 2019), we also used extended DFDs but detected violations of RBAC policies, which supports the AC analysis of the running example. The previous approaches are data-driven but none claims and evaluates detecting both, IF and AC, violations within the same model. We could not find such an approach.

5 MODELING PRIMITIVES FOR CONFIDENTIALITY

This section covers the unified meta model. We mine modeling primitives from identified domain concepts of two related approaches. A modeling language integrates these modeling primitives to represent confidentiality concepts. After that, we describe how to integrate these modeling primitives into existing ADLs.

Mining Modeling Primitives. To find shared and specific modeling primitives for representing IF and AC at design level, we compare the primitives of the recent IF approach SecDFD (Tuma et al., 2019) and our recent AC approach DDSA (Seifermann et al., 2019). Both formalize modeling primitives by meta models. The upper part of Figure 2 describes the simplified SecDFD meta model, the lower part describes

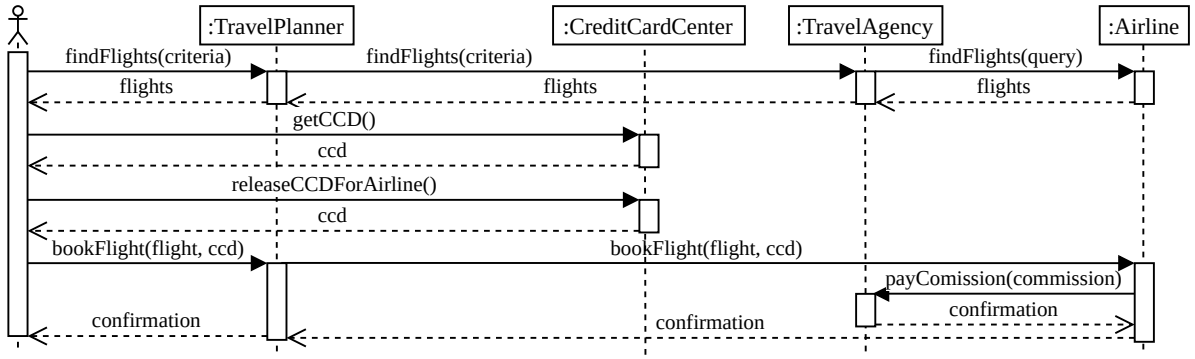

Figure 1: Interactions of components during the booking of a flight in the TravelPlanner running example.

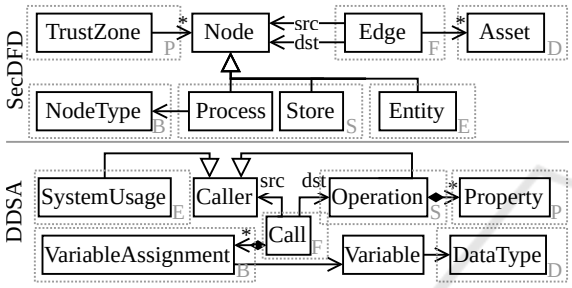

Figure 2: Meta models of SecDFD (top) / DDSA (bottom).

the simplified DDSA meta model. Here and for the remainder of this paper, simplification means that we omit associations, attributes, classes or interfaces not essential for our explanations. We identified the following six concepts shared between both meta models that we marked by upper-case letters in Figure 2.

System Structure (S): is about describing interacting parts of systems. SecDFD describes the structure by processes and stores, which are basic elements of DFDs. DDSA builds the structure on services realized by operations. The corresponding analyses treat these differently named elements roughly the same way.

External Entities (E): are users or external systems that can receive information from the system or send information to it. SecDFD and DDSA represent these entities in the same way but using different names.

Flows (F): transmit information between the previously described elements. SecDFD also calls this flows. Even if DDSA describes flows by the control flow term "call", both elements describe the same concept and are treated the same way in analyses.

Properties (P): describe characteristics of entities. SecDFD allows properties on all nodes within and outside of the system but limits the property to a trust or attack zone. DDSA only supports characterizing operations but allows characteristics to be finite value sets. Analyses use both concepts to compare characteristics of nodes with other or fixed expected values.

Data (D): describes the exchanged information. SecDFD describes the type of exchanged information by assets. DDSA uses variables and data types for the same. Both concepts only specify types but no particular characteristics such as a level 3 classification.

Behavior (B): describes the influence of system elements on characteristics of exchanged information. Both approaches use analyses based on label propagation, so the behaviors are given in terms of label propagation functions. SecDFD couples the label propagation function with the node type. DDSA does the same but the underlying model supports defining behaviors in the model by means of assignments.

To summarize, both approaches represent the same fundamental modeling concepts by different modeling primitives. SecDFD introduces modeling primitives tailored for particular analyses. DDSA incorporates a flexible approach requiring designers to model properties and behavior specific to intended analyses. Both analyses use label propagation with different sets of labels and propagation functions.

Unified Modeling Primitives. Based on the previous comparison, we assume that we can analyze IF and AC within a unified modeling approach that subsumes the previously mined primitives. We do not discuss or evaluate usability as part of this section or paper. Tuma et al. (Tuma et al., 2020) already demonstrated that a modeling language based on data flows and appropriate tooling is, depending on the particular use case, usable. Our modeling primitives are also based on data flows and we can also provide a catalogue of predefined behaviors, labels and analysis definitions. Therefore, we do not see this as a research question anymore. Instead, we aim for determining how we can represent IF and AC analyses within one modeling language that complies with DFD terminology. Therefore, we define a new meta model that incorporates all modeling primitives and supports flex-

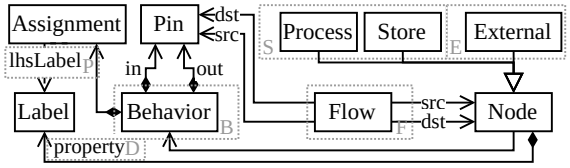

Figure 3: Meta model of unified modeling primitives.

ible analysis definitions. We stick to the data flow terminology used by SecDFD and pick up the idea of flexible property and characteristic definitions of DDSA. Additionally, we provide means for specifying reusable behaviors and properties. Figure 3 illustrates the simplified resulting meta model.

The meta model distinguishes nodes and edges. Nodes are the commonly used modeling primitives for defining the system structure *S* (processes and stores) and external entities *E*. Every node has a defined behavior *B* specified by a label propagation function and required inputs and outputs via pins. Assignments specify the label propagation function by assigning a truth value to a label on an output pin based on constants, logical expressions and references to labels on input pins. A label with a false truth value means that it is not applied. Pins decouple behaviors from particular nodes and incoming flows. Therefore, pins enable reusing behaviors in multiple nodes. A flow *F* transports all labels of an output pin of a source node to the input pin of a destination node. To represent properties *P* of nodes, we assign labels to nodes.

Integration of Modeling Primitives in ADLs. To show how existing ADLs can make use of our modeling primitives and analyses to be defined later, we describe the integration with the ADL Palladio. Palladio supports data flows and control flows, so we show the integration with both paradigms. The integration always consists of two steps. First, we identify modeling primitives that have no counterparts and that have to be added to the ADL, therefore. Second, we define a transformation for this extended ADL to our ADL-independent meta model shown in Figure 3. The transformation allows us to reuse the same analysis definitions for all possible ADLs. We exemplify these steps for Palladio by going through the modeling primitives, identifying their existing counterparts and describing the transformation to the modeling primitives. These transformations are tailored to the ADL. To increase comprehensibility, we explain the general ideas of the transformations but avoid low level descriptions. The full transformations are available in our data set (Seifermann et al., 2021).

Control Flow (CF) Palladio: already provides modeling concepts that fit our modeling primitives:

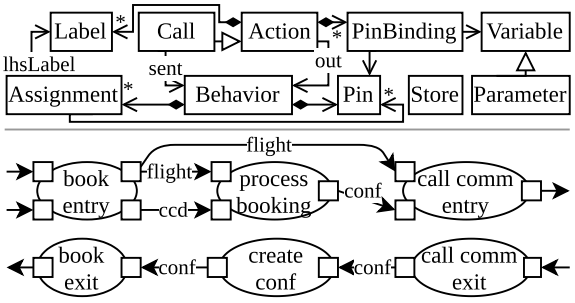

Figure 4: Integration into control-oriented Palladio (upper) and transformation result for flight booking service (lower).

Processes are represented by services. Flows are represented by calls between services. Pins are represented by parameters and return values. Each of these elements can be transformed one by one. Palladio already considers users, which match external entities. The modeling primitives often missing in ADLs are stores, labels and behaviors. We extend Palladio by stores as special types of components. The label and behavior primitives can be integrated as they are. An overview on the extended meta model is given in the upper part of Figure 4. Most of the elements can be easily transformed. The only exception are services, which require one process (entry) for receiving and distributing all input parameters and one process (exit) for returning output parameters. The transformation result for the flight booking service in the lower part of Figure 4 illustrates this.

Data Flow (DF) Palladio: already provides modeling concepts that fit our modeling primitives: Processes are represented by data channels. Flows are represented by data flows between data channels. Pins are represented by the outgoing and incoming data flows. Each of these elements can be transformed one by one. Palladio already considers users, which matches external entities. The modeling primitives often missing in ADLs are stores, labels and behaviors. We extend Palladio by stores as special type of data channel. The label and behavior primitives can be integrated as they are. Figure 5 gives an overview on the extended meta model. Most of the elements can be easily transformed. Figure 5 shows the transformation result for the flight booking service. There are less processes compared to the CF version because entry and exit processes are not needed.

6 ANALYSIS DEFINITION

Confidentiality analyses are defined on the previously presented unified modeling primitives. Therefore, the

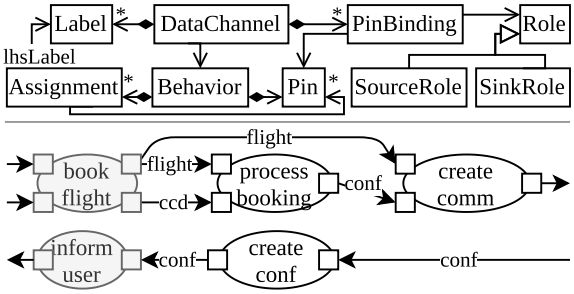

Figure 5: Integration into data-oriented Palladio (upper) and transformation result for flight booking service (lower).

definition can be reused over all ADL integrations. Both, SecDFD and DDSA use label propagation to derive labels of transmitted data based on initial labels and label propagation functions. In the running example, an initial label is required for the criteria passed to the travel planner. By applying label propagation functions of traversed processes, the analysis derives the label of criteria or query data passed through the system. After deriving all data labels, the analysis compares data labels with node labels. In the RBAC running example, the analysis compares access rights labels on data with role labels on nodes. A violation occurs if a node accesses data with an access rights set not containing the role of the node. To summarize, we need a) data labels, b) node labels, c) behaviors (i.e. label propagation functions) and d) the label comparison function for an analysis definition.

We provide analysis definitions of common IF and AC analyses as presented by IFlow (Katkalov, 2017), DDSA (Seifermann et al., 2019) and SecDFD (Tuma et al., 2019). We also define analyses for the AC models DAC, MAC, ABAC and for Taint-based Memory Protection via Access Control (TMAC) (Wang et al., 2009) that combines IF and AC.

Two behaviors are used in all analyses. We briefly introduce them here and do not mention them anymore later: The *forward* behavior copies all labels of the input to the output pin. The *create* behavior has no inputs and sets labels on the output pin explicitly.

Non-interference Considering High/Low (2L). IF analyses of SecDFD (Tuma et al., 2019) compare the trust zone of nodes with data classifications. An IF policy is violated if nodes within attack zones access data classified as high. This can also be seen as taint analysis. We need two classification labels (high/low) applied to data and two zone labels (attack/trusted) applied to nodes. In addition, we need two labels (high/low) to represent the content of encrypted data. Encryption and decryption can be used to transport data classified high through attack zones. The used behaviors are as follows: The *encrypt* behavior uses incoming classification labels as encrypted content labels on outputs and sets the classification to low. The *decrypt* behavior uses incoming encrypted content labels as classification label on outputs. The *join* behavior takes multiple inputs and sets the encrypted content and the classification label to high if at least one input has a high label for the respective label type.

Non-interference with Hierarchical Lattice (HL). IF analyses of IFlow (Katkalov, 2017) for the cases *TravelPlanner*, *ContactSMSManager* and *DistanceTracker* compare the node clearance with data classifications. An IF policy is violated if nodes with a certain clearance level access data with higher classification level. This analysis type is called non-interference using a hierarchical lattice. The clearance and classification levels are hierarchical security levels building a lattice. There is one label per level that can be attached to data or nodes. The used behaviors are as follows: The *sync* behavior acts like the forward behavior but has an additional input pin, whose labels are ignored. The *declassify* behavior acts like the forward behavior but replaces labels of a certain type with new fixed labels. For instance, the declassification behavior replaces the classification level *User* of the credit card data with the new classification level *User,Airline* in the IF running example. The *join* behavior applies the most restrictive classification label of all inputs to the output.

Non-interference with Lattice Groups (LG). The *PrivateTaxi* case of IFlow (Katkalov, 2017) requires that nodes owned by a certain role must not access certain critical data types. More precisely, a distance calculation service must not access contact data of users and a taxi broker service must not access route data. A violation means one of these rules is violated. This can be seen as two separated lattice groups containing two levels each. The lattice of the first group is non critical data followed by route data. The lattice of the second group is non critical data followed by contact data. The analysis definition is tailored to the particular use case. For the sake of simplicity, we omit detailed descriptions for this case. Informally said, we have labels representing critical data types and labels representing services. A violation occurs if a forbidden combination of service label and critical data type label exists at one node. The data set (Seifermann et al., 2021) provides further details.

Core Role-based Access Control (RBAC). The DDSA AC analysis (Seifermann et al., 2019) covers Core RBAC (Furnell, 2008, pp. 71) that defines permissions of roles and role assignments. The analysis

reports a node holding no role that is contained in the set of allowed roles of incoming data. The particular data and node labels depend on the used roles. In the running example, the labels are *user*, *travel agency* and *airline*. A violation is reported if the intersection of data and node labels is empty. There are two behavior types. The *intersect* behavior applies the intersection of all input labels to the output pins. The *declassify* behavior acts like the forwarding behavior but additionally adds a specified role to the outputs.

Discretionary Access Control (DAC). DAC (Furnell, 2008, pp. 61) directly assigns access permissions (subject and allowed action) to objects. In DFDs, read and write are feasible actions to consider. We define permission labels that apply to stores because they are the best to detect read and write actions. A permission label is an element of the cross product of available subjects and actions (read/write). External entities have an identification label. Traversal labels on data represent the nodes that already have been traversed. The read action analysis determines the originating store of received data by the traversal labels. The analysis reports a violation if there is no read permission label matching the identity label of the external entity. The analysis for write violations looks at the traversal labels of data received by stores to identify external entities writing to the store. The analysis reports a violation if there is no write permission label matching the identity label of the external entity. The only used behavior takes inputs and applies the union of the traversal labels including the label of the current node to all outputs.

Mandatory Access Control (MAC). MAC (Furnell, 2008, pp. 64) provides general policies without referring to particular nodes or data. One prominent MAC security model is the military security model, which reduces to the already explained noninterference with hierarchical lattice.

Attribute-based Access Control (ABAC). ABAC (Furnell, 2008, pp. 74) defines attribute-based subject and object selectors and assigns permissions between them. All attributes become labels that are assigned to nodes or data. An analysis selects nodes and data based on their labels and reports a violation for any undefined combination. Because ABAC is highly flexible, we do not give a specific analysis definition.

Taint-based Memory Protection via Access Control (TMAC). TMAC (Wang et al., 2009) extends AC for computer memory with restrictions based on taint labels. The analysis description depends on the used AC model, so we can use previously presented analysis descriptions and extend them by taint analysis. In the simplest form, the taint label is just a single label (data is tainted). Additionally, there is a label for critical nodes. Existing AC behaviors are extended to consider taint labels: the taint label is applied to all outputs if there is at least one tainted input. An additional *declassification* behavior removes the taint label. The AC label comparison is extended to ensure that tainted data never arrives at critical nodes.

7 EVALUATION

We evaluate the expressiveness of presented modeling primitives (C1) and the accuracy of the analysis definitions (C2). We describe evaluation goals and metrics in Section 7.1. The evaluation design to achieve the goals is given in Section 7.2. Section 7.3 covers the cases of our case study-based evaluation. We present and discuss evaluation results in Section 7.4. Section 7.5 and Section 7.6 discuss threats to validity and limitations. We briefly report on the implementation and data availability in Section 7.7.

7.1 Evaluation Goals and Metrics

We structure our evaluation by the Goal-Question-Metric (GQM) approach (Basili et al., 1994): we formulate evaluation goals that we can achieve by defining evaluation questions. Metrics are used to answer the questions. Our evaluation goals are:

G1) Evaluate the expressiveness of the unified model including its ADL integration (C1). This is useful because other approaches cannot represent both, IF and AC, together.

G2) Evaluate the accuracy of the proposed analyses (C2). Analysis results have to be accurate to be useful, which would not be the case for high rates of false positives or false negatives.

Both evaluation goals support each other because an expressive model is only useful if it can serve its purpose, which is detecting confidentiality violations automatically here. Natural language is a counterexample, because it is expressive but does not support automated analyses. Contrarily, accurate analysis results are only useful if a considerable amount of systems and confidentiality mechanisms are supported.

We evaluate the expressiveness of the unified model (G1) by evaluating its integration into the Palladio ADL because this is the intended way to use it. The ADL does not impose more restrictions than the modeling primitives because the transformation from

the ADL can produce all modeling primitives. Therefore, it is not necessary to evaluate the expressiveness twice. The corresponding evaluation questions are as follows: Q1.1) Can the extended ADL express behavior relevant for common IF analyses? Q1.2) Can the extended ADL express behavior relevant for common AC analyses? Q1.3) Can the extended ADL express behavior relevant for combined IF and AC analyses? The case study described in Section 7.2 answers the questions by determining the share of successfully modeled systems and analyses.

We evaluate the accuracy of the analyses (G2) by the following evaluation questions: Q2.1) Can the IF analyses provide sufficiently accurate results? Q2.2) Can the AC analyses provide sufficiently accurate results? Q2.3) Can the combined IF and AC analyses provide sufficiently accurate results? To answer the questions, we execute the analyses modeled for G1. We compare the results with expected previously defined results or results from related work if available. Based on the comparison, we calculate the recall (also called sensitivity) $r = \frac{t_p}{t_p+f_n}$ with true positives t_p and false negatives f_n as well as specificity $s = \frac{t_n}{t_n+f_p}$ with true negatives t_n and false positives f_p. Both metrics are commonly used in medicine for rating binary classifications (Metz, 1978). Additionally, we calculate precision $p = \frac{t_p}{t_p+f_p}$, which is commonly used in rating the accuracy in information retrieval.

7.2 Evaluation Design

Our evaluation is based on a case study, which is a common way of evaluating modeling notations and analyses in the field of security (Nguyen et al., 2015; van den Berghe et al., 2017). The procedure is the same for every case. A case is about one particular system, modeled by one particular paradigm (CF/DF) and about one particular confidentiality analysis. For every case, we create two system variants: One variant contains no issue, so the automated analysis shall not report a violation. The other variant contains an issue, so the automated analysis shall report a violation. We inject the same issue as reported in related work. If none has been reported, we build an issue based on common mistakes such as wrong calls, wrong wiring of components and so on. We model both variants using the extended Palladio ADL, define the analysis in terms of our modeling primitives as described in Section 6 and execute the analysis. After the last step, we can classify the produced artifacts, i.e. models and analysis results, and collect the metrics to answer the evaluation questions.

To calculate the metric for questions Q1.1, Q1.2 and Q1.3, we test if we can model variants and corresponding analyses and compare that to the total amount of variants and analyses. We classify a variant as successfully modeled if i) we could represent all labels and behaviors described for the particular analysis in Section 6 and ii) we could formulate the label comparison function for the particular case. For each question, we calculate the share of supported variants.

To calculate the metrics for answering questions Q2.1, Q2.2 and Q2.3, we classify the analysis results. The ground truth for expected analysis results is the classification of the variant during its creation by us, which in turn is justified by related work whenever available. We classify each reported violation individually but aggregate these individual classifications for calculating the metric. This means the analysis result of each variant contributes exactly one true/false positive/negative to the metric calculation. Thereby, we avoid that a large amount of good results for one particular analysis definition hides a small number of bad results for another analysis definition. If the analysis reports a violation for a variant not containing an issue, we classify the result as false positive f_p. If the analysis does not report a violation for a variant containing an issue, we classify the result as false negative f_n. If the analysis does not report a violation for a variant not containing an issue, we classify the result as true negative t_n. For every violation reported for a variant containing an issue, we check that every reported violation traces back to the actual issue. If there is at least one violation that does not trace back to the actual issue, we classify the result as f_p. Otherwise, the result becomes a true positive t_p.

We consider cases about IF, AC and their combination. Additionally, we aim for validating the support of the CF and the DF paradigm as we claim in this paper. Therefore, we must have at least one case per element of the cross product of the confidentiality mechanism (IF, AC and combination) and the system behavior paradigm (CF and DF). To achieve this, we select cases from the related approaches IFlow (Katkalov, 2017) (IF), SecDFD (Tuma et al., 2019) (IF), DDSA (Seifermann et al., 2019) (AC) or create own cases to fill the remaining gaps (further AC analyses and combination with IF). We group cases of related approaches sharing the same analysis definition into equivalence classes. We select one case of each class because modeling further cases of this class would not give us any more insight into expressiveness or accuracy. If the case is only available in one paradigm (CF or DF), we derive a case in the other paradigm based on the available descriptions. An overview on the selected cases is given in Table 1. Every value in a cell represents a case and refers to the analysis definition applied in the case. The selected

Table 1: Cases considered for case study evaluation. CF means Control Flow, DF means Data Flow.

System	Inform. Flow CF	Inform. Flow DF	Access Control CF	Access Control DF
TravelPlanner	HL	HL	RBAC	RBAC
DistanceTracker	HL	HL	RBAC	RBAC
PrivateTaxi	LG	LG		
BankingApp	–	–		
JPmail	2L	2L		
ImageSharing			DAC	DAC
FlightControl			MAC	MAC
BankBranches			ABAC	ABAC

cases cover all analyses that we claimed to support: non-interference with hierarchical lattice (*HL*), lattice groups (*LG*) and high/low (*2L*), as well as the access control models. In addition, we realized TMAC in the TravelPlanner for CF and DF. We describe the case selection and the cases in Section 7.3.

7.3 Case Selection and Description

First, we describe the equivalence classes of cases of the related approaches (Katkalov, 2017; Tuma et al., 2019; Seifermann et al., 2019). We describe the selected cases afterwards. IFlow (Katkalov, 2017) provides five cases. *TravelPlanner*, *DistanceTracker* and *ContactSMS* target non-interference with hierarchical lattices (HL). *PrivateTaxi* targets non-interference with lattice groups (LG). *BankingApp* targets non-interference between users. We select *PrivateTaxi*, *BankingApp*, and *TravelPlanner* as representative cases for each class. We add *DistanceTracker* because we already used *TravelPlanner* as running example. SecDFD (Tuma et al., 2019) provides five cases targeting non-interference with high/low lattice (2L). All cases are equivalent, so we select *JPmail*. DDSA (Seifermann et al., 2019) provides the cases *TravelPlanner*, *DistanceTracker* and *ContactSMS* known from IFlow but tailored to RBAC. All cases are equivalent, so we select *TravelPlanner* and *DistanceTracker*. We could not identify cases including reference results for DAC, MAC, ABAC. So, we defined cases based on descriptions of the access control models (Furnell, 2008, pp. 61). *ImageSharing* covers DAC, *FlightControl* covers MAC and *BankBranches* covers ABAC. Further, we extend the travel planner example by TMAC (Wang et al., 2009).

TravelPlanner has already been explained in Section 3. With respect to TMAC, we extend the analysis to report data coming from an external into the system that is not immediately validated by the system. *DistanceTracker* consists of a tracking service on a user device that collects GPS locations, calculates a distance and submits the distance to a distance tracking service. The distance tracking service must never have access to plain locations. *PrivateTaxi* is a broker platform to bring together drivers and riders going in the same direction. The broker must never know the route or location of users but delegates proximity calculations to a trusted third party service. *BankingApp* is a system to withdraw money after authentication. Different users must never interfere with each other. *JPmail* is an email system. Sender and receiver encrypt their communication to not allow intermediate nodes to read the email body. Intermediate nodes must never access the plain mail body. *ImageSharing* is a file sharing system to share family photos with multiple users. The store has access permissions attached: Parents can write images and read them. Relatives can read images. Indexing bots such as used by search engines must not access images. The analysis tests if illegal access occurs. *FlightControl* is a flight monitoring system covering weather monitoring, civil flight monitoring and military flight monitoring. Weather staff must only access weather information. Civil flight staff can access civil plane locations in addition. Military flight staff can access military plane locations in addition. The analysis tests if subjects access information with a classification higher than their clearance. *BankBranches* is a banking system to manage accounts and calculate credit lines spanning two regions. Employees can manage regular customers within their region. Managers can manage customers from all regions and celebrities. The analysis tests these restrictions based on subject and data attributes.

The smallest case has 4 structural elements (components, data channels, interfaces) and one behavior. The two largest case have 34 structural elements and 8 types of behaviors. We consider the cases, especially those taken from related work, realistic.

7.4 Evaluation Results and Discussion

We structure the section by the evaluation goals and questions. We start with the expressiveness goal G1.

To answer Q1.1, we tried to model the cases, i.e. the system and the analysis definition, *TravelPlanner*, *DistanceTracker*, *PrivateTaxi*, *BankingApp* and *JPmail* with the CF and DF architectural description language integration, which sums up to ten cases and twenty variants. We successfully modeled all systems and analysis definitions except the *BankingApp*, which means a coverage of 80%. We could not model the banking case because the case is about isolating tenants, i.e. the corresponding analysis requires dis-

tinguishing different actors and data that are the same on a type level. We did not focus on expressing entities and data on instance level because this leads to fine-grained models not appropriate for early design phases. If required, our ADL integration can express tenants by scenario-based modeling, which represents tenants as dedicated new actor types and system parts used by tenants as dedicated new system parts. However, this comes with additional modeling effort and obfuscates the intended architecture.

To answer Q1.2, we tried to model *TravelPlanner*, *DistanceTracker*, *ImageSharing*, *FlightControl* and *BankBranches* with the CF and DF architectural description language integration, which are ten cases and twenty variants. We could successfully model all cases and variants, which leads to full coverage.

To answer Q1.3, we tried to model the TMAC-extended *TravelPlanner* with the CF and DF architectural description language integration, which are two cases and four variants. We could successfully model all cases and variants, which leads to full coverage.

To summarize, we could cover all tested AC and combined cases, as well as most of the IF cases. We confirmed a previously known and intended limitation with respect to representing tenants in systems. Therefore, we achieved good expressiveness with respect to most of the common AC and IF analyses.

The accuracy evaluation (G2) is divided in IF, AC and combined cases. To answer Q2.1, we run analyses about non-interference with hierarchical lattice on *TravelPlanner* and *DistanceTracker*, non-interference with lattice groups on *PrivateTaxi*, as well as non-interference considering high/low on *JPmail*. All cases are realized with the CF and DF architectural description language integration, which sums up to eight cases and sixteen variants. The analysis correctly reported no error on variants without issue and only reported valid violations on variants with issue. This means precision, recall and specificity are 1.

To answer Q2.2, we run analyses for DAC on *ImageSharing*, for MAC on *FlightControl*, for ABAC on *BankBranches* and for RBAC on *TravelPlanner* and *DistanceTracker*. We realized all cases in the CF and DF architectural description language integration, which sums up to ten cases and twenty variants. The analyses correctly reported no error on variants without issue and only reported valid violations on variants with issue. This means precision, recall and specificity are 1.

To answer Q2.3, we run the TMAC analysis on *TravelPlanner* with the CF and DF architectural description language integration. The analysis correctly reported no error on the variant without issue and only reported valid violations on the variant with issue. This means precision, recall and specificity are 1.

To summarize, we achieved perfect accuracy on all cases that we could express in G1. This shows that we can achieve the same results as related approaches but support a wider range of analyses. Additionally, it validates that our analysis concepts and their implementation are sufficient to achieve valuable results.

7.5 Threats to Validity

We structure the discussion of threats to validity by the four categories of validity for case study research by Runeson et al. (Runeson et al., 2012, pp. 71).

Construct validity requires that measures taken actually measure what the researcher had in mind. The measures are the share of expressible variants and the accuracy of our analysis definitions. The share of expressible variants is feasible because we do not have unbalanced data sets, i.e. more analyses of a certain type than another. However, the share does not allow in-depth analysis of reasons for limited expressiveness. Therefore, we additionally discussed the results and possible causes. The accuracy metrics are established for rating classifiers.

Internal Validity: requires that investigated factors are actual factors influencing the results. For expressiveness, we investigate the factors modeling language and analysis definition. However, there are additional factors: Limited experience in the modeling language or security mechanisms can influence results negatively. We can exclude this factor because the person that modeled the systems and defined the analyses is an author of this paper and therefore has sufficient experience with both. Too simple scenarios can positively influence results. We consider this threat low because the cases stem from related work or cover the core concepts of the corresponding confidentiality mechanisms. For accuracy, we investigate the factors of analysis definitions and the analysis execution concepts. An additional factor are wrong result classifications. To cope with that, we ensured that reported violations trace back to the actual issue.

External Validity: requires that generalizations of results are valid. In research based on case studies, there is no statistical representative sample. Therefore, generalizations are only valid for cases with comparable characteristics. Our results can be generalized for ADLs that provide means for describing system structure and behavior by calls or data exchange. Also, we think that we can support further confidentiality analyses that use type-level information rather than instance-level information.

Reliability: requires that the results are independent of a particular researcher. The modeling cer-

tainly depends on the experience of the executing researcher. Other researchers might not achieve the same expressiveness results. However, this does not invalidate the results because expressiveness is about the upper bound of expressible scenarios and not about what average users can achieve. The latter would be subject to a usability evaluation, which we did neither do nor focus on. It is sufficient to understand that the scenarios can be modelled by our ADL extension by reviewing the models in our data set (Seifermann et al., 2021). With this data set, researchers can also replicate the accuracy study. Analysis execution and metrics can be done objectively.

7.6 Limitations

Our proposed approach has two major limitations. First, analyses are restricted to type level. Because we focus on software architectures and early software designs, we chose to omit instance level information such as specific data values or complex interplay between two actors of the same type. We doubt that such detailed information is actually available in early design phases and certainly modeling this in detail would increase the modeling effort.

Second, we cannot cover implicit flows (King et al., 2008), i.e. information flows via control flow dependencies. Our approach, like most other model-based approaches, requires explicit flows. Implicit flows are often subtle and require detailed models or source code. Again, this would certainly increase the modeling effort considerably.

7.7 Tool Support and Data Availability

We realized a prototype within the Palladio tooling. We realized all meta models described previously in the Eclipse Modeling Framework (EMF). We integrated editing support in existing Palladio editors to ease modeling. To reduce the modeling effort, we integrated means for reusing analysis definitions including behaviors, labels and label comparisons. We plan to provide a catalogue containing labels, behaviors and analysis definitions of the analyses presented in this paper except for the case-specific ones such as ABAC. An automated model transformation maps the Palladio model to an instance of our generic meta model. Another automated model transformation maps the generic model instance to a logic program doing the label propagation. All parts related to the logic program are an engineering task, so we did not present them in this paper. A DSL (Hahner et al., 2021) simplifies specifications of new analyses.

All artifacts of our prototype are publicly available and also part of the data set (Seifermann et al., 2021). We include all information to replicate the evaluation such as implementation artifacts, modeled cases and analysis results including their classification.

8 CONCLUSIONS

In this paper, we presented i) a unified model for representing systems including their behavior relevant for IF and AC confidentiality analyses and ii) definitions for common IF and AC analyses. To the best of our knowledge, no data-oriented design time approach can represent both within one modeling language. To derive the model, we compared one IF and one AC approach to identify a set of shared information between both analysis types as well as analysis-specific model elements. Based on the derived unified model, we defined eight common IF and AC analyses. We evaluated expressiveness and accuracy in a case study. We evaluated 22 cases and experienced satisfying expressiveness and accuracy.

The benefit of the approach is twofold. First, it is now possible to consider IF and AC within the same architecture. Architects can start modeling shared aspects including system structure and rough behavior and decide later on the confidentiality mechanism. Without our approach, architects had to decide in the beginning and had to remodel large parts of the architecture when switching between mechanisms or define mappings between the IF and the AC modeling approach. Second, we explicitly considered how existing ADLs can be extended to support the confidentiality analyses we proposed. We demonstrated this by our ADL integration into Palladio. The integration into an existing ADL can lower the learning effort because architects only have to learn the new constructs.

In future, we plan to evaluate the ADL integration approach for more ADLs and to find ways to analyze tenants without instance level information.

ACKNOWLEDGEMENTS

The DFG (German Research Foundation) – project number 432576552, HE8596/1-1 (FluidTrust) and the KASTEL institutional funding supported this work.

REFERENCES

Abdellatif, T. et al. (2011). Automating information flow control in component-based distributed systems. In *CBSE'11*, pages 73–82.

Abi-Antoun, M. et al. (2007). Checking threat modeling data flow diagrams for implementation conformance and security. In *ASE'07*, pages 393–396.

Alabiso, B. (1988). Transformation of Data Flow Analysis Models to Object Oriented Design. In *OOPSLA'88*, pages 335–354.

Almorsy, M. et al. (2013). Automated software architecture security risk analysis using formalized signatures. In *ICSE'13*, pages 662–671.

Basili, V. R. et al. (1994). The Goal Question Metric Approach. In *Encyclopedia of Software Engineering - 2 Volume Set*, pages 528–532.

Berger, B. J. et al. (2016). Automatically Extracting Threats from Extended Data Flow Diagrams. In *ESSoS'16*, volume 9639, pages 56–71.

Boehm, B. and Basili, V. R. (2001). Software Defect Reduction Top 10 List. *Computer*, 34(1):135–137.

DeMarco, T. (1979). *Structured analysis and system specification*. Prentice-Hall, Englewood Cliffs, N.J.

Furnell, S., editor (2008). *Securing information and communications systems: principles, technologies, and applications*. Artech House, Boston.

Hahner, S. et al. (2021). Modeling data flow constraints for design-time confidentiality analyses. In *ICSA 2021*. accepted, to appear.

Hedin, D. et al. (2012). A Perspective on Information-Flow Control. In *Software Safety and Security - Tools for Analysis and Verification*, volume 33 of *NATO Science for Peace and Security Series - D*, pages 319–347.

Heyman, T. et al. (2012). Reusable Formal Models for Secure Software Architectures. In *WICSA'12*, pages 41–50.

Hoisl, B. et al. (2014). Modeling and enforcing secure object flows in process-driven SOAs: an integrated model-driven approach. *SoSym*, 13(2):513–548.

Hoo, K. S. et al. (2001). Tangible ROI through Secure Software Engineering. *Secure Business Quart.*, 1(2):1–3.

International Organization for Standardization (2018). ISO/IEC 27000:2018(E). Standard, ISO.

Jilani, A. et al. (2011). Comparative Study on DFD to UML Diagrams Transformations. *WCSIT*, 1(1):10–16.

Jürjens, J. (2005). *Secure Systems Development with UML*. Springer-Verlag, Berlin Heidelberg.

Katkalov, K. (2017). *Ein modellgetriebener Ansatz zur Entwicklung informationsflusssicherer Systeme*. PhD Thesis, University of Augsburg, Augsburg. German.

Katkalov, K. et al. (2013). Model-Driven Development of Information Flow-Secure Systems with IFlow. In *SocialCom'2013*, pages 51–56.

Khedker, U. et al. (2009). *Data Flow Analysis: Theory and Practice*. CRC Press, Inc., USA, 1st edition.

King, D. et al. (2008). Implicit Flows: Can't Live with 'Em, Can't Live without 'Em. In *ICISS*, pages 56–70.

Lowry, E. S. and Medlock, C. W. (1969). Object code optimization. *Communications of the ACM*, 12(1):13–22.

McGraw, G. (2006). *Software Security - Building Security In*. Addison-Wesley Professional.

Metz, C. E. (1978). Basic principles of ROC analysis. *Seminars in Nuclear Medicine*, 8(4):283–298.

Microsoft Corporation et al. (2009). Microsoft SDL: Return-on-Investment. https://www.nccgroup.trust/globalassets/our-research/us/whitepapers/isec-partners—microsoft-sdl-return-on-investment.pdf. accessed 20/11/25.

Nguyen, P. H. et al. (2015). An extensive systematic review on the Model-Driven Development of secure systems. *IST*, 68:62–81.

OMG (2020). Unified Architecture Framework. Standard formal/19-11-07, Object Management Group.

Reussner, R. H. et al. (2016). *Modeling and Simulating Software Architectures - The Palladio Approach*. MIT Press, Cambridge, MA.

Runeson, P. et al. (2012). *Case Study Research in Software Engineering: Guidelines and Examples*. Wiley.

Sandhu, R. S. et al. (1994). Access control: principle and practice. *IEEE ComMag*, 32(9):40–48.

Seifermann, S. et al. (2019). Data-Driven Software Architecture for Analyzing Confidentiality. In *ICSA'19*, pages 1–10.

Seifermann, S. et al. (2021). SECRYPT 2021 Evaluation Data Set. https://doi.org/10.5281/zenodo.4699417.

Shostack, A. (2014). *Threat modeling: designing for security*. Wiley, Indianapolis, IN.

Shull, F. et al. (2002). What we have learned about fighting defects. In *METRICS'02*, pages 249–258.

Sion, L. et al. (2018). Solution-aware data flow diagrams for security threat modeling. In *SAC'18*, pages 1425–1432.

Sion, L. et al. (2020). Security Threat Modeling: Are Data Flow Diagrams Enough? In *ICSEW'20*.

Smith, G. (2007). Principles of Secure Information Flow Analysis. In *Malware Detection*, ADIS, pages 291–307.

Torres, W. et al. (2020). A systematic literature review of cross-domain model consistency checking by model management tools. *SoSym*.

Tuma, K. et al. (2019). Flaws in Flows: Unveiling Design Flaws via Information Flow Analysis. In *ICSA'19*, pages 191–200.

Tuma, K. et al. (2020). Automating the early detection of security design flaws. In *MoDELS'20*, pages 332–342.

van den Berghe, A. et al. (2017). A Model for Provably Secure Software Design. In *FormaliSE'17*, pages 3–9.

van den Berghe, A. et al. (2017). Design notations for secure software: a systematic literature review. *SoSym*, 16(3):809–831.

Venson, E. et al. (2019). Costing Secure Software Development: A Systematic Mapping Study. In *ARES'19*, pages 1–11.

Wang, L. et al. (2009). TMAC: Taint-Based Memory Protection via Access Control. In *DEPEND'09*, pages 19–27.

Werle, D. et al. (2020). Data Stream Operations as First-Class Entities in Component-Based Performance Models. In *ECSA'20*, pages 148–164.

Xu, W. et al. (2006). Taint-Enhanced Policy Enforcement: A Practical Approach to Defeat a Wide Range of Attacks. In *USENIX Security Symposium*.

Zdancewic, S. (2004). Challenges for Information-flow Security. In *PLID'04*, page 5.