Detection Strategies for Microservice Security Tactics

Uwe Zdun, *Member, IEEE,* Pierre-Jean Queval, Georg Simhandl, Riccardo Scandariato, Somik Chakravarty, Marjan Jelić, and Aleksandar Jovanović

Abstract—Microservice architectures are widely used today to implement distributed systems. Securing microservice architectures is challenging because of their polyglot nature, continuous evolution, and various security concerns relevant to such architectures. This article proposes a novel, model-based approach providing detection strategies to address the automated detection of security tactics (or patterns and best practices) in a given microservice architecture decomposition model. Our novel detection strategies are metrics-based rules that decide conformance to a security recommendation based on a statistical predictor. The proposed approach models this recommendation using Architectural Design Decisions (ADDs). We apply our approach for four different security-related ADDs on access management, traffic control, and avoiding plaintext sensitive data in the context of microservice systems. We then apply our approach to a model data set of 10 open-source microservice systems and 20 variants of those systems. Our results are detection strategies showing a very low bias, a very high correlation, and a low prediction error in our model data set.

Index Terms—Microservices, Microservice Architecture Security, Detection Strategies, Metrics, Conformance Checking.

1 Introduction

Microservice architectures consist of independently deployable, scalable, and changeable services [1], [2], [3]. Key characteristics are development in independent teams, polyglot technology stacks, cloud-native technologies and architectures, lightweight containers, loosely-coupled service dependencies, end-to-end tracing and monitoring, and continuous delivery [1], [2], [4]. This article focuses on the security aspects of microservice architectures. Despite numerous published guidelines and best practices [5], [6], [7], architecting microservice systems is challenging concerning security. This is due to the size and complexity of microservice systems, the many relevant security concerns in such systems, their polyglot nature, and the need for continuous evolution and frequent release of these systems. In this context, manually validating whether numerous required security features are used as intended throughout the system is a time-consuming and error-prone task. Architectural abstraction can help focus only on the relevant aspects of architecturally significant security features. However, substantial effort is still required, e.g., to check a large-scale system's architecture for conformance to security recommendations.

This article presents an approach for checking the conformance to recommendations on security-related Architectural Design Decisions (ADDs) via detection strategies. The conformance relation is generally defined as the consistency between models [8]. This concerns the relation between a software system's architecture and its intended architecture [9].

A *Detection Strategy* is defined as *"the quantifiable expression of a rule by which design fragments that conform to that rule can be detected."* [10]. To enable the formulation of concrete detection strategies for conformance relations, we defined four exemplary ADDs with security tactics as decision options. Further, we specified metrics representing the different options of the ADDs. We define these for several security aspects not yet modeled by ADDs or metrics in the literature, namely access management, traffic control, and avoidance of sensitive plaintext data. Then we define our detection strategies as rules on top of the metrics. In contrast to prior work on detection strategies [10], we do not base our approach on simple data filters only. Instead, we use a statistical analysis based on ordinal logistic regression to derive prediction models, which we then use to construct our detection strategies. This article aims to study the following research questions:

- **RQ1.** How can we automatically detect conformance to recommendations on ADDs and tactics on security in microservice architecture models?
- **RQ2.** How well does this detection perform?

This work is based on a dataset of 10 open-source microservice systems that we have manually modeled in our previous work [11], [12] and that are partially (i.e., 3 of the 10 systems) automatically extracted from the source code. We have added 20 variants in which possible violations of ADD options or refactorings for improvement are introduced based on the discussions in the relevant literature. In addition to the cases, our prior work provides (1) a method for automatically extracting decomposition models for polyglot microservice systems from the source code [12] and (2) an approach for metrics detection in such models [11]. These building blocks of our approach are only

- *U. Zdun, P. Queval, and G. Simhandl are with the University of Vienna, Faculty of Computer Science, Research Group Software Architecture, Vienna, Austria.*
  E-mail: {firstname.lastname}@univie.ac.at
- *R. Scandariato is with the Hamburg University of Technology (TUHH), Hamburg, Germany.*
- *S. Chakravarty, M. Jelic. A. Jovanovic are with the European Risk and Resilience Institute (EU-VRi), Stuttgart, Germany.*

Manuscript received ...; revised ...

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2023.3276487

briefly introduced in this article. Our novel contributions, which we will focus on, are (1) a new detection strategy approach, (2) a novel set of ADDs and security tactics used to validate our approach, and (3) a substantially extended formalization for the models and metrics. We have established a ground truth based on a manual assessment by five industrial experts. We compare the detection strategies statistically to the ground truth to evaluate our approach. Our results show that for each of the four ADDs, we found at least one detection strategy that uses a regression model with very low bias, has low or very low prediction error and a very high prediction correlation with ground truth data. Our approach requires manual assessment and modeling for creating the dataset and a regression model. Still, once a fitting regression model has been established, the approach can be applied automatically, e.g., as a component in an analysis tool (see Section 6) or as an automated step in a continuous delivery pipeline.

As an additional contribution, this article provides a validation of the study results provided in [11]. That is, the conformance detection approach introduced in [11], which is used by our novel detection strategies, is replicated here based on an entirely different set of ADDs, a new formalization approach, an entirely new metrics set, a new recommendation/ground truth analysis study and new security extensions in our model data set.

This article is structured as follows: First, in Section 2, we overview our approach and discuss the research methods used. Then, Section 3 presents the ADDs for microservice security tactics considered in this article and the ground truth derived from them. Section 4 formally specifies the metrics and detection strategies. Next, Section 5 presents the analysis of the regression models and the derived detection strategies. Section 6 describes an industrial resilience assessment tool in which we have applied our approach and the lessons learned. Section 7 discusses our findings and potential threats to validity. Then we compare to related work in Section 8, and in Section 9, we conclude.

2 Overview and Research Methods

Figure 1 illustrates our approach in use. A user either models a microservice system as a software decomposition (or component & connector) model with security annotations (as specified in Section 4) or automatically extracts such a model (e.g., using the automatic code extraction approach from our prior work [12]). The automatic extraction can usually be run repeatedly, e.g., in the context of a continuous delivery pipeline, without the need for additional manual work. Once such as model is in place, our prototype can automatically detect all relevant metrics values. These provide the necessary data for running the detection strategies to detect conformance to ADD-based recommendations. The ordinal logistic regression models are part of the detection strategies and require the metrics as input. They are derived from a representative model data set like the one contributed in this paper and can be fine-tuned by, e.g., adding more or different models to the data set, if our model data is inappropriate for a given use of our approach (e.g., because the system under investigation's domain, size,

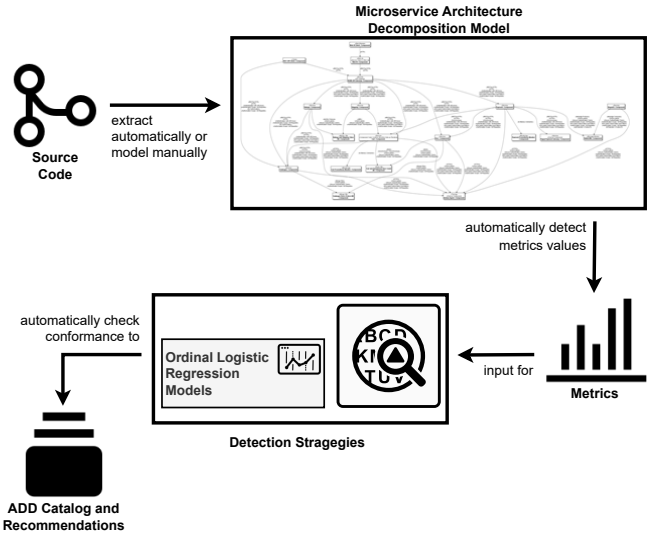


Fig. 1: Approach in Use

or technology concepts are substantially different to the systems in our model data set).

Our research started with a **data collection and analysis** in which we have studied existing microservice-specific recommendations by industry organizations such as NIST [5], OWASP [6], or the Cloud Security Alliance [7]. Before they got involved in this article, a team of industrial security experts, including the last three authors of this study, independently analyzed these recommendations. In addition, within the AssureMOSS EU project[1], the author team conducted a multi-vocal literature study (i.e., scientific and grey literature) to confirm the findings.

The authors **derived a catalog of ADDs** with security tactics as decision options from this data. We selected 4 of these ADDs that the industrial security experts and the industry recommendations judge as highly relevant for microservice systems, covering different aspects. By purpose, we selected an entirely different set of ADDs than used in our prior works [11] – this way, this article provides another validation of the approach in [11].

The authors then studied 10 open source microservices systems as **case studies** line-by-line and manually annotated each security feature in their source code (all are published on GitHub, see Table 1). To enable us to study the continuous evolution of these systems with a focus on the security ADDs, we developed 20 variants of the systems in which possible violations of ADD options or refactorings for improvement are introduced, based on the discussions in the relevant literature. Apart from the specific variations described for the variants in Column "Main Security Features/Issues" Table 1, all other system aspects remained stable. This is shown in an excerpt of Model PM0 in Figures 2 and 3. Figure 3 highlights the changes compared to PM0 in red, described in the Figure caption. We assume that our evaluation systems are, or reflect, real-world practical examples of microservice architectures. Many of them are open-source systems realized by practitioners to demonstrate practices or technologies, and thus they are at

1. https://assuremoss.eu/en/

most of medium size complexity. An essential feature of our dataset to reflect current microservice practices is that the systems are highly polyglot, using many different programming languages and technologies. As is common in real-life systems, in the dataset, the required information is located in many different kinds of files, such as programs and scripts in various programming and scripting languages and configuration files of many different technologies (see Column "Programming Languages and Technologies Used" in Table 1).

We used our existing CodeableModels tool[2], a Python implementation for **precisely specifying meta-models, models, and model instances** in code. Based on CodeableModels, we have automated **code generators** to generate graphical visualizations of all meta-models and models in PlantUML. We have also realized **detectors** to find all relevant aspects of the metrics in the models and the **automatic calculation of metrics**.

This modeling step was done manually in our work for many of the case study systems. It is also possible to automate this step: Our existing static code analysis approach for architecture reconstruction of polyglot microservice systems [12] can be applied here. Due to the polyglot nature of microservice systems, this requires a modest initial specification effort, though. Models for the systems RS0 and ES0 have been automatically reconstructed using this approach in our prior work (see [12]). PM0 has been automatically reconstructed, too (not published in prior work).

We then performed a **systematic assessment on support or violation of the collected security tactics**. The three industrial security experts in the author team plus two additional industrial security experts (from the company SEARCH-LAB) independently derived a recommendation based on the results of our tactics study. The result provides informal guidance for security experts to judge systems such as those in our models manually. Next, the other authors applied this recommendation as an ordinal rating scheme to each model variant summarized in Table 1 to create a ground truth for our study. Then the five industrial security experts reviewed the rating scheme and the ratings in the ground truth. In case of inconsistency of the votes, we performed a discussion among the involved experts to resolve the conflict. We would have applied a majority vote if the debate would not yield consistent votes, but the experts reached a consensus after the discussion in all cases.

Independently of the work on the ground truth, on simple example cases, we **developed our detection strategies**. To this end, we first developed a set of metrics that automatically decide each decision point in our ADDs. These metrics are formally defined in Section 4. Next, our statistical analysis assessed how well the hypothesized metrics could predict the ground truth data by performing an ordinal regression analysis. Ordinal regression is widely used for modeling an ordinal response's dependence on independent predictors applicable in various domains. For the ordinal regression analysis, we used the *lrm* function from the *rms* package in R [13], [14].

The authors then used the ordinal regression models to construct two possible detection strategies for each recommendation on the ADDs provided by the industrial experts. The detection strategies use the regression model's means and fitted prediction methods as their basis [14]. We **compare and evaluate** the resulting detection strategies' performances for our model data set using the Mean Square Error (MSE) and Spearman correlation.

The resulting model data set, the model code, and the statistical evaluations are provided as an open-source data set to enable the replicability of our study[3].

3 ADDs for Microservice Security Tactics and Ground Truth Assessment

In this section, we describe the four ADDs containing microservice security tactics that are studied in this article. Next, we describe the manual analysis of these ADDs to establish a ground truth for our study.

3.1 ADD: Access Management/Backend Authorization (BE_AU)

When considering access management in the context of microservice architecture decomposition models, we mainly found various authorization-related tactics. It is important to note that authorization is crucial for all parts of a microservice architecture, but especially for services reachable directly or indirectly from the clients. Thus, we treat the two decision scopes, backend authorization, and authorization from the clients/UIs, in two separate ADDs that offer the same decision options.

The following **decision options** (security tactics) can be chosen for Backend Authorization (BE_AU):

- **Token-based Authorization:** Authorization is performed using a cryptographic access token issued by a central access management server, such as an OAuth 2.0 token.
- **Encrypted Authorization Information:** Some other kind of encrypted authorization scheme is used, but not with a standardized central access management server.
- **Plaintext Authorization Information:** Authorization information is transferred as plaintext.
- **Plaintext-based Authorization Information over an Encrypted Protocol:** Authorization information is transferred as plaintext over a secure (i.e., encrypted) communication protocol such as TLS/SSL.
- **No Authorization:** No authorization method is used, but authorization is required. That is, the fact that authorization is not provided is a security flaw in the system.
- **Authorization Not Required:** The connector does not need any form of authorization, and the fact that it is missing is not a security flaw. This is, for instance, the case if access is allowed for each identified client or in public APIs with no access restrictions.

2. https://github.com/uzdun/CodeableModels

3. We will publish the data set as an open access data set on the long-term archive Zenodo upon publication of this article. For the time of the review, we provide it as an anonymously accessible link at https://ucloud.univie.ac.at/index.php/s/ROOCWYH5fpS1d7w, Password: PtKbCZLn29cNkAA

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2023.3276487

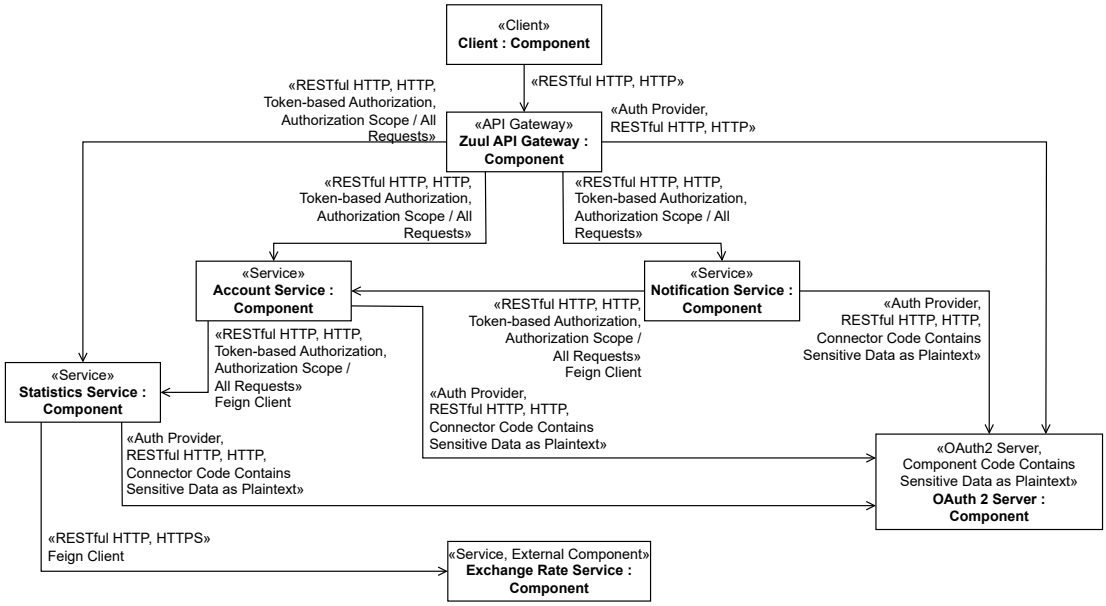


Fig. 2: Excerpt of the Model PM0 (7 out of 16 components and their connectors) showing service interactions, API Gateway, and OAuth2 server. Only the excerpt of the necessary stereotypes is shown for clarity.

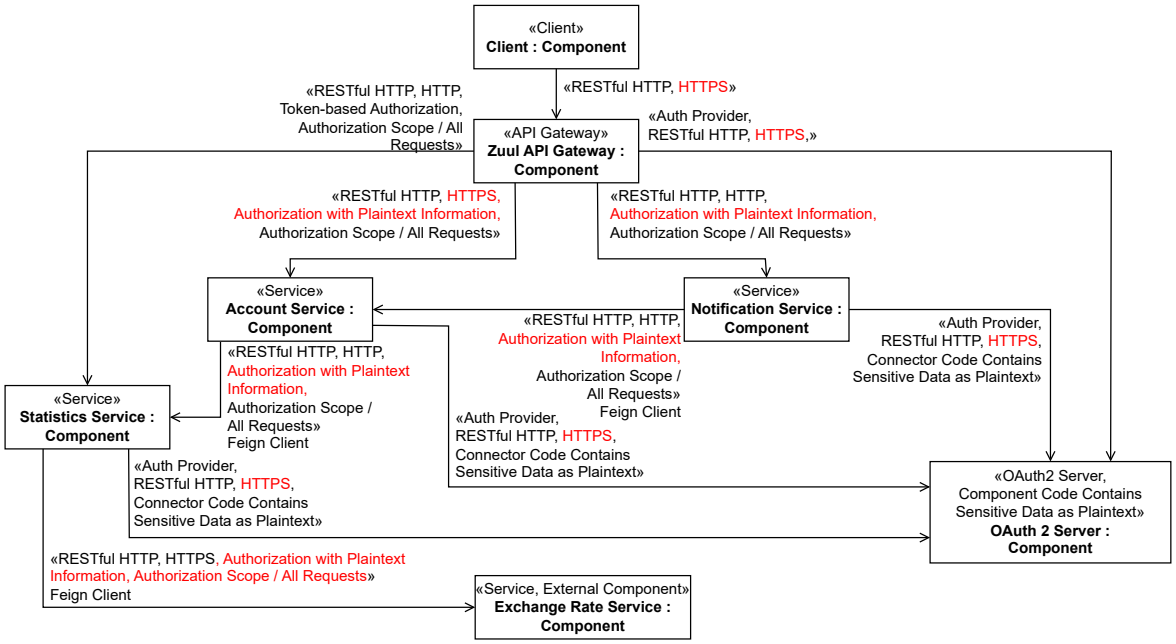


Fig. 3: Excerpt of the Model PM1 as an example to show changes in a variant. Changes highlighted in red: The variant introduces a security flaw as it only uses limited plaintext authorization but fixes some issues regarding encrypted communication by using HTTPS on several connections. The same traffic control and sensitive data issues are present in the excerpt as in the PM0 excerpt.

ID	Number Components	Number Connectors	Domain	Main Security Features/Issues	Programming Languages and Technologies Used	Source (URL)
AC0	7	10	Account management	Some OAuth2 support (client-service), API gateway, some sensitive data issues	Java, Spring Boot, Maven, MySQL, JDBC, OAuth2, Zuul, Eureka, Feign, JPA	https://github.com/piomin/sample-spring-oauth2-microservices/tree/with_database
AC1	8	11		Only plaintext authorization, no API gateway, substantial sensitive data issues		
AC2	8	14		Token-based auth. (backend/client-service), some traffic control, small sensitive data issues		
BA0	11	17	Bank accounts and money transfer	API-gateway based traffic control, no authorization, substantial sensitive data issues	Java, Scala, Groovy, Javascript, Spring Boot, Gradle, Swagger, Docker, Docker Compose, Eventuate, SQL, JDBC, Shell Scripts	https://github.com/cer/event-sourcing-examples
BA1	12	21		Some authorization, some traffic control, substantial sensitive data issues		
BA2	12	24		Limited authorization, some traffic control, substantial sensitive data issues		
CI0	8	12	Cinema booking service	API-gateway based traffic control, no authorization, small sensitive data issues	JavaScript, Express, SPDY, HTTP Proxy, Morgan, Helmet, Docker, Docker Swarm, NPM, RAML, Mongo DB, SSL, Shell Scripts	https://github.com/Crizstian/cinema-microservice
CI1	12	21		Limited authorization, some traffic control, small sensitive data issues		
CI2	12	23		Plaintext/secure connection authorization, some traffic control, small sensitive data issues		
CO0	13	16	Store management	No authorization and traffic control, sensitive data issues	Java, Maven, Glassfish, JBoss, JavaEE, EJB, Java Servlets, JSF, JAX-RS, JDBC, Shell Scripts	https://github.com/cocome-community-case-study/cocome-cloud-jee-microservices-rest
CO1	12	19		Limited authorization, some traffic control, small sensitive data issues		
CO2	12	16		Plaintext/secure connection authorization, limited traffic control, small sensitive data issues		
EP0	11	11	Enterprise planner	API gateway, no authorization, some sensitive data issues	C#, ASP.NET, Docker, Docker Compose, JWT, ASP.NET Core Auth, Linq, Sql Server, Shell/Powershell Scripts	https://github.com/gfawcett22/EnterprisePlannert
EP1	16	22		Limited authorization, some traffic control, some sensitive data issues		
EP2	16	23		Token-based auth. (backend/client-service), limited traffic control, small sensitive data issues		
ES0	18	29	eShop reference application	Backends-for-Frontends gateways, some authorization, some sensitive data issues	C#, Javascript, ASP.NET, Open API, ASP.NET Core Auth, Web-SPA, WebMVC, SQL Server, K8S, Linq, Envoy, Azure, ELK, Github Actions, Powershell Scr.	https://github.com/dotnet-architecture/eShopOnContainers
ES1	18	37		Limited authorization, some traffic control, some sensitive data issues		
ES2	18	35		Token-based auth. (backend/client-service), limited traffic control, small sensitive data issues		
OB0	13	25	Online boutique application	Backends-for-Frontends gateways, no authorization, no sensitive data issues	Java, grpc, Gradle, Protobuf, Kubernetes, Docker, Helm, Istio, Kustomize, Terraform, Skaffold, Cloud Build, Shell Scripts	https://github.com/GoogleCloudPlatform/microservices-demo
OB1	13	23		Some encrypted/plaintext authorization, some traffic control, small sensitive data issues		
OB2	13	30		Limited encrypted/plaintext authorization, some traffic control, small sensitive data issues		
PM0	16	37	Microservice metrics collecting	API Gateway, OAuth2 (client-service/some backend), substantial sensitive data issues	Java, Spring Boot, Maven, Docker, Docker Compose, Travis CI, Turbine, MongoDB, Feign, Eureka, OAuth2, Zuul	https://github.com/sqshq/piggymetrics
PM1	18	40		Limited plaintext auth. (many over secure conn.), some traffic control, some sensitive data issues		
PM2	17	34		Plaintext auth. over secure connections, some traffic control, substantial sensitive data issues		
RS0	19	30	Robot shop application	API Gateway, no authorization, no sensitive data issues, no encryption	Javascript, Go, PHP, Java, Feign, Express, DCOS, OpenShift, K8S, Docker, Dock. Swarm/Compose, MongoDB, Redis, RabbitMQ, MySQL, Shell Scripts	https://github.com/instana/robot-shop
RS1	19	32		Limited encrypted authorization, limited traffic control, some sensitive data issues		
RS2	19	34		Plaintext authorization/secure connections, some traffic control, some sensitive data issues		
TE0	17	26	Tap-And-Eat application	No traffic control, no authorization, no sensitive data issues	Java, Spring Boot, Docker, Docker Compose, Maven, MySQL, Shell Scripts	https://github.com/jferrater/Tap-And-Eat-MicroServices
TE1	17	28		Limited plaintext auth. (backend/client-service), no traffic control, no sensitive data issues		
TE2	17	30		Limited plaintext authorization (mainly client-service), no traffic control, no sensitive data issues		

TABLE 1: Overview of modeled systems (size, details, and sources)

These can be decided for each occurrence of the following **decision context:** *Each connector between two components in the system (such as system services, databases, infrastructure components, discovery services, or access management servers), but not connections to clients and UIs (or between them) and/or external services. To be decided for each connector.*

3.2 ADD: Authorization on Paths from Clients or UIs to System Services (CP_AU)

For CP_AU, the same **decision options** (security tactics) can be chosen as for Backend Authorization (BE_AU) but in a different decision context. While BE_AU is concerned with each backend connection in the microservice architecture (e.g., service to service or service to database), CP_AU is concerned with the paths between clients/UIs and system services (direct connections or propagated connections along the paths between them). That is, CP_AU can be decided for each occurrence of the following **decision context:** *Each direct or transitive connector between a client or UI to a system service. In this context, transitive means that the connector can cross API Gateways and similar frontend components first, and then other system services, but no other kinds of components (such as infrastructure services, databases, and so on). To be decided for each connector.*

3.3 ADD: Traffic Control (TC)

When considering Traffic Control in our scope of architecture decomposition models, mainly different kinds of *Facades* [15] that shield system services from direct access are discussed as solutions in the security recommendations [5], [7]. Such a *Facade* acts as a reverse proxy and routes requests from clients to backend services. It also realizes cross-cutting concerns such as authentication, authorization, SSL termination, and monitoring to support security tasks [5]. The most common pattern for this is the *API Gateway* [3], but there are also variants and homegrown solutions.

The following **decision options** (security tactics) can be chosen for Traffic Control (TC):

- **API Gateway** [3] provides a single endpoint for the clients and internally maps the requests to backend microservices.
- **Backends for Frontends** [3] is a variation of API Gateway that defines a separate gateway for each kind of client, e.g., a Web app, a mobile app, and a public API gateway.
- **Frontend Service:** While the prior options usually use dedicated technologies for establishing traffic control, some systems use a homegrown frontend service that acts as a Facade [15] for other services in the system. It can only offer the traffic control features built into that service.
- **Direct Access from Clients to Services:** Clients access system services directly, and thus no traffic control is provided.

These can be decided for each occurrence of the following **decision context:** *Each possible path from a client or a UI to a system service. To be decided for each such path.*

3.4 ADD: Avoiding Plaintext Sensitive Data (SD)

Sensitive data in plaintext should not be used anywhere in a system system [5], [7]. Instead, encrypted solutions and keys should be used. In architecture decomposition models, components can contain plaintext sensitive data, such as a service or database storing user passwords. The interactions (or connectors) between components can use plaintext sensitive data, e.g., to transfer unencrypted credentials over the wire.

The following **decision options** (security tactics) can be chosen for Avoiding Plaintext Sensitive Data (SD):

- **Avoiding Plaintext Sensitive Data in Components** means storing no secrets in components or using encryption methods to secure them properly. This tactic requires a systematic investigation of the data that is classified as sensitive.
- **Avoiding Plaintext Sensitive Data in Connectors** means to not use plaintext secrets in the interactions realized by connectors, mainly distributed ones. Again, encryption, e.g., of the connection and maybe local storage of the secret, is needed to implement this tactic. This tactic requires a systematic investigation of the data that is classified as sensitive.
- **Plaintext Sensitive Data in Components** means a specific component stores sensitive data in plaintext form. This should usually be avoided.
- **Plaintext Sensitive Data in Connectors** means a specific connector uses or stores sensitive data in plaintext form. This should usually be avoided.

These can be decided for each occurrence of the following **decision context:** *Each component and connector in the model. To be decided for each such model element.*

3.5 Recommendations and Ground Truth Assessment

To establish a ground truth for evaluating conformance to the ADDs described in the previous sections, the three industrial security experts on the author team first worked with other experts in their organizations to create recommendations based on the results of our tactics study (i.e., from security guidelines, gray literature, and scientific literature studies). The other authors then analyzed these recommendations, compared them to actual implementations in the case study systems, and selected the recommendations that were the focus of our ADDs. The results are the recommendations per ADD below, where more or less preferred ADD options (tactics) are mapped on a 5-point ordinal scale: $++$: very well supported; $+$: well supported, but aspects of the solution could be improved; $\sim$: serious flaws in security design, but significant support is already found in the system; $-$: serious flaws in security design, but initial support can already be found in the system; $--$: no support for the security tactic can be found in the system. The authors then discussed this evaluation scheme again with the three industrial security experts until a consensus was reached. The other authors then evaluated the 30 cases for conformance to each of the ADDs. The ratings were again reviewed by the three security assessment experts on the author team. In addition, two industrial security experts from another company reviewed our models, metrics, and code.

Some parts of the recommendations below result in a unique score, especially the extreme cases ($++$, $--$) often refer to unique quantities such as *all connectors* or *no connectors*. However, some other values contain fuzzy statements such as *the vast majority of connectors*, where human judgment is required depending on the system model to decide how large the set must be to be acceptable in that particular model. For example, system context, system size, and the system's domain can lead to individually different judgments for different models.

The resulting recommendation scheme for **Backend Authorization (BE_AU)** is:

- $++$: All distributed backend connectors are authorized with **Token-based Authorization** provided by a central access management server.
- $+$: All distributed backend connectors are authorized with a **Token-based Authorization**, or some other kind of **Encrypted Authorization Information**, or with **Plaintext-based Authorization Information over an Encrypted Protocol**, and not all are authorized with **Token-based Authorization**.
- $\sim$: Either the large majority of distributed backend connectors is authorized with **Token-based Authorization**, **Encrypted Authorization Information**, or **Plaintext-based Authorization Information over an**

Encrypted Protocol; or all distributed backend connectors are authorized, but some or all of those are authorized using **Plaintext Authorization Information**.

- $-$: At least some distributed backend connectors are authorized, but either **Plaintext Authorization Information** is used and not all connectors are authorized; or, if no **Plaintext-based Authorization** is used, less than the large majority of distributed backend connectors is authorized with **Token-based Authorization**, **Encrypted Authorization Information**, or **Plaintext-based Authorization Information over an Encrypted Protocol**.
- $--$: No distributed backend connectors are authorized.

If the *Authorization Not Required* option is selected, the connector should not be further analyzed with regard to access management aspects.

The recommendation scheme for **Authorization on Paths from Clients/UIs to Services (CP_AU)** is exactly the same scheme as in the Backend Authorization, but not for the context of distributed backend connectors, but for the scope of paths from clients or UIs to system services.

The recommendation scheme for **Traffic Control (TC)** is:

- $++$: All possible paths from a client/UI to a system service are passing through a dedicated gateway solution such as an **API Gateway** or **Backends for Frontends**.
- $+$: All possible paths from a client/UI to a system service are passing through a dedicated **API Gateway** or **Backends for Frontends**, or through some kind of **Frontend Service**.
- $\sim$: The large majority of the possible paths from a client/UI to a system service are passing through a dedicated **API Gateway** or **Backends for Frontends**, or through some kind of **Frontend Service**.
- $-$: At least some possible paths from a client/UI to a system service are passing through a dedicated **API Gateway** or **Backends for Frontends**, or through some kind of **Frontend Service**.
- $--$: No **API Gateways**, **Backends for Frontends**, or **Frontend Service** are found on the possible paths from a client/UI to a system service.

The recommendation scheme for **Avoiding Plaintext Sensitive Data (SD)** is:

- $+$: No **component** and no **connector** contains, uses, or stores plaintext sensitive data.
- $\sim$: Almost all of the **components** and **connectors** contain, use, and store no plaintext sensitive data.
- $-$: The large majority of **components** and **connectors** contain, use, and store no plaintext sensitive data.
- $--$: Less than a large majority of **components** and **connectors** contain, use, and store no plaintext sensitive data.

The "$++$" recommendation is not used in SD, as no well-supported but not optimal option exists.

Based on the recommendations, the ground truth assessment in Table 2 was derived. That is, Table 2 lists the ground truth assessments for each decision and for each of the case study systems from Table 1.

4 Detection Strategies, Models, and Metrics Specification

This section describes metrics for measuring conformance to the common microservice security tactics described as decision options in Section 3. Our metrics are based on a microservices-based architecture decomposition model. For a complete formal definition of this model, see [11], [16]. We only present the necessary model elements and extensions used in this article.

4.1 Basic Architecture Decomposition Model

Formally, an architecture decomposition model M is a tuple $(CP_M, CN_M, CPT_M, CNT_M, cn\_source, cn\_target, cp\_type, cn\_type)$ where:

- CP_M is a finite set of **component nodes** in Model M.
- $CN_M \subseteq CP_M \times CP_M$ is an ordered finite set of **connector edges**.
- CPT_M is a set of **component types**.
- CNT_M is a set of **connector types**.
- $cn\_source : CN_M \rightarrow CP_M$ is a function returning the component that is the **source** of a link between two components.
- $cn\_target : CN_M \rightarrow CP_M$ is a function returning the component that is the **target** of a link between two components.
- $cp\_type : CP_M \rightarrow \mathbb{P}(CPT_M)$ is a function that maps each component to its set of **direct and transitive component types** (for a formal definition of component types and type hierarchies see [11], [16]).
- $cn\_type : CN_M \rightarrow \mathbb{P}(CNT_M)$ is a function that maps each connector to its set of **direct and transitive connector types** (for a formal definition of component types and type hierarchies see [11], [16]).

Below, to simplify the metrics definition texts, when we simply say Component cp or Connector cn is of type t, we refer to the use of the function call $cp\_type(cp)$ or $cn\_type(cn)$, respectively. Figures 2 and 3 show example models from the PM case modeled using the UML.

4.2 Component and Connector Types

We distinguish various component and connector types introduced in the text below where they are needed. The full type hierarchies are modeled in the CodeableModels distribution[4]. It uses microservice architecture component types such as *Service*, *API_Gateway*, *Database*, *Monitoring*, etc. Microservice decompositions have many different kinds of connector types between these components, such as *RESTful HTTP*, *HTTPS*, *JDBC*, etc., to denote the kind of interactions between the components.

Based on this, we define in this article some security-specific extensions such as $TokenBasedAuthorization$,

4. The component type hierarchy can be found at https://github.com/uzdun/CodeableModels/blob/master/docs/_images/Component_Stereotypes.png, and the connector type hierarchy is at https://github.com/uzdun/CodeableModels/blob/master/docs/_images/Connector_Stereotypes.png. Both models are explained in the documentation of CodeableModels: https://uzdun.github.io/CodeableModels/07_meta_model_with_stereotypes.html.

Decision	AC0	AC1	AC2	BA0	BA1	BA2	CI0	CI1	CI2	CO0	CO1	CO2	EP0	EP1	EP2	ES0	ES1	ES2	OB0	OB1	OB2	PM0	PM1	PM2	RS0	RS1	RS2	TE0	TE1	TE2
BE_AU	-	-	++	--	-	∼	--	∼	+	--	-	+	--	+	++	-	∼	+	--	∼	+	-	∼	+	--	∼	+	--	∼	∼
CP_AU	++	-	++	--	∼	∼	--	∼	+	--	∼	+	--	+	++	-	-	+	--	-	+	++	∼	+	--	∼	+	--	∼	-
TC	++	--	-	++	∼	∼	++	+	+	--	-	∼	++	∼	+	++	-	∼	+	-	-	++	+	+	++	∼	-	--	--	--
SD	-	--	∼	--	--	--	-	-	-	--	-	-	-	-	∼	-	-	-	+	∼	∼	--	-	--	+	∼	∼	+	+	+

TABLE 2: Ground Truth Assessment for the Case Study Systems

$EncryptedAuthorizationInformation$, $Authorization\text{-}WithPlaintextInformation$, and so on. They represent security-specific information extracted from the code and used for model annotation. Figures 2 and 3 show example models from the PM case with the component and connector types being rendered as stereotypes.

Below we use these types to define type-selection functions. For instance, in the backend authorization metrics below the $distributed\_backend\_connectors\_requiring\_authorization : \mathbb{P}(CN_M) \rightarrow \mathbb{P}(CN_M)$ function is used as the basis for calculation and common divisor. It is defined as:

$$distributed\_backend\_connectors\_requiring\_authorization(cn) = connectors\_that\_require\_authorization(distributed\_backend\_connectors(cn))$$

The function is essentially a cascade of type selections. $distributed\_backend\_connectors$ first selects the connectors which are neither connecting to $Client$ nor UI components to get all backend connectors. And then, the distributed backend connectors are the subset of these connectors, which are not of type $InMemConnector$ (i.e., in-memory connectors). The function $connectors\_that\_require\_authenthorization : \mathbb{P}(CN_M) \rightarrow \mathbb{P}(CN_M)$ selects the connectors that are not of the type $AuthorizationNotRequired$. This type indicates connections that explicitly should not get authorized, e.g., because they are offered in a public API.

4.3 Paths

A basic notion in a number of the metrics below is a **path**: A path p is a sequence of components $p = (c_1, \ldots, c_n)$ with $c_1 \ldots c_n \in CP_M$ which are all connected via connectors such that $\forall c_n, c_{n+1} \in P \exists cn \in CN_M : cn\_source(cn) = c_n \wedge cn\_target(cn) = c_{n+1}$. Let P_M denote the set of all paths in model M.

The function $all\_paths\_from\_clients\_or\_uis\_to\_system\_services : \mathbb{P}(CP_M) \rightarrow \mathbb{P}(P_M)$ selects all paths from clients or UIs to system services. The function first selects all components of type $Client$ or UI as clients and all components of type $Service$ as system services. A service is a **system service** if it is not of the type $ExternalComponent$ (i.e., no external services are system services), $MiddlewareService$ (i.e., no middleware infrastructure services such as a Discovery Service), or $Facade$ (i.e., no frontend services or gateways with the sole purpose of shielding the system from clients). Then the function uses a simple Depth-First Search algorithm to calculate all paths from clients to services. From those paths, we select only the ones that are well-formed in the sense that first $Clients$ or UIs are on the paths, then zero, one, or more $Facades$ (e.g., $APIGateways$ or frontend services are of type $Facade$), and finally one or more system services (as defined above). Paths going across other components such as $Databases$ or $MiddlewareServices$ are excluded; paths going into the system, then out of the system, and back into the system are also excluded, too.

The function $client\_service\_path\_connectors\_requiring\_authorization : \mathbb{P}(CP_M) \rightarrow \mathbb{P}(CN_M)$ is based on this function. It first selects the connectors from the result of $all\_paths\_from\_clients\_or\_uis\_to\_system\_services$ using another function $connectors\_on\_client\_service\_paths : \mathbb{P}(P_M) \rightarrow \mathbb{P}(CN_M)$. This function returns the set of all connectors on a set of paths (without connectors having $Facades$ or $ExternalComponents$ as targets or that are of type $InMemoryConnector$). Then, it selects the connectors that require authentication using the function $connectors\_that\_require\_authorization$.

4.4 Detectors

Detectors are functions that calculate **Detector Results** DR, such as $detector : \mathbb{P}(ME\_P_M) \rightarrow \mathbb{P}(DR)$. ME_M are model elements of a model M, with: $\forall CN_M, CP_M \in M : ME_M \supset CP_M \wedge ME_M \supset CN_M$. The detectors either work on such model elements or on paths: $ME\_P_M = ME_M \cup P_M$. DR is a tuple (mp, res) with $mp \in ME\_P_M$ and $res \in \{successful, undefined, failed\}$.

The function $d\_success : \mathbb{P}(DR) \rightarrow \mathbb{P}(DR)$ selects only the successful detection results in a detector result set, $d\_fail : \mathbb{P}(DR) \rightarrow \mathbb{P}(DR)$ the failed ones, and $d\_undefined : \mathbb{P}(DR) \rightarrow \mathbb{P}(DR)$ the undefined ones. The function $d\_elements : \mathbb{P}(DR) \rightarrow ME_M$ returns the model elements contained in a detector result set.

In a set of model elements and paths $\supset ME\_P_M$, let the function $components : \mathbb{P}(ME\_P_M) \rightarrow \mathbb{P}(CP_M)$ select the components in the set, $connectors : \mathbb{P}(ME\_P_M) \rightarrow \mathbb{P}(CN_M)$ the connectors in the set, and $paths : \mathbb{P}(ME\_P_M) \rightarrow \mathbb{P}(P_M)$ the paths in the set.

4.5 Metrics Definitions

The formal metrics definitions are provided in Table 3. All metrics have values ranging from 0 to 1, with 1 indicating full support and 0 indicating no support.

The first compartment in the table provides metrics on backend authorization. They are all based on the $distributed\_backend\_connectors\_requiring\_authorization$ function explained above. The first metric AUB detects general support for authorization in the backend without considering the used type of authorization. The following metrics AUB_T, AUB_E, AUB_P, and AUB_C are calculating a similar ratio but only for specific types of authorization, or any authorization over an encrypted, secure connection. Finally, AUB_A combines the authorization methods that are considered to be secure enough, i.e. AUB_T, AUB_E, and AUB_C in one metric.

The second compartment in the table provides metrics on authorization on client/service paths. They are all based on the $client\_service\_path\_connectors\_requiring\_authorization$ function, which uses components as input and delivers a connector set. Apart from that, the AUC metrics are constructed in the same fashion as the AUB metrics.

In the third compartment of the table, we see metrics on traffic control. Firstly, the superior method of using gateways is measured in the GWP metric, and then FEP measures the acceptable practice of realizing traffic control with frontend services. Finally, GFP measures the ratio of both practices in the paths, leading to different results than just looking at the two individual metrics GFP and FEP, e.g., as both practices could be applied on a path. GFP, however, can be biased as an implicit weight is introduced (the number of gateway vs. frontend services).

Finally, in the fourth compartment, the plaintext sensitive data use is measured, first for the components in CMP, then for the connectors in CNP, and finally for both in CCP. It makes sense to have CMP and CNP in addition to CCP, as the number of components and connectors are introducing an implicit weight in the metric, which can potentially bias the results of CCP.

4.6 Detection Strategies

Based on our metrics, we define two alternative detection strategies that use the mean and fitted prediction methods for ordinal logistic regression, as provided, e.g., in R's *rms* package [14]. First, we define the *Means Predictor Detection Strategy*. Let OR_{add} denote an ordinal regression model for the ADD add (with $add \in \{BE\_AU, CP\_AU, TC, SD\}$, i.e. it is in the set of all ADDs), $predict_{mean}$ the prediction function using the means prediction method, and $m \in M$ the input model for the prediction. The mean prediction computes the estimated mean Y by summing values of Y multiplied by the estimated probabilities $P(Y = j)$ [14]. Then, $means\_predictor\_detection\_strategy : OR_{add} \times M \rightarrow \{`++', `+', `\sim', `-', `--'\}$ is a detection strategy function computing the levels ++, +, $\sim$, and - of our ground truth scheme from Section 3.5:

$$means\_predictor\_detection\_strategy(OR_{add}, m) = \begin{cases} `++' : & if\ predict_{mean}(OR_{add}, m) \geq int(`++') - 0.5 \\ `+' : & if\ predict_{mean}(OR_{add}, m) \geq int(v + `') - 0.5 \\ `\sim' : & if\ predict_{mean}(OR_{add}, m) \geq int(`\sim') - 0.5 \\ `-' : & if\ predict_{mean}(OR_{add}, m) \geq int(`-') - 0.5 \\ `--' : & otherwise \end{cases}$$

The *Fitted Predictor Detection Strategy* function $fitted\_predictor\_detection\_strategy : OR_{add} \times M \rightarrow \{`++', `+', `\sim', `-', `--'\}$ is based on the fitted prediction method, which gets all the individual probabilities $Y = j$ [14]. We define $level\_of\_max$ as a function that first selects the maximum probability in the vector returned by $predict_{fitted}$. Then it gets the ordinal level ++, +, $\sim$, -, – of the vector value with the maximum probability:

$$fitted\_predictor\_detection\_strategy(OR_{add}, m) = \begin{cases} `++' : & if\ level\_of\_max(predict_{fitted}(OR_{add}, m)) = `++' \\ `+' : & if\ level\_of\_max(predict_{fitted}(OR_{add}, m)) = `+' \\ `\sim' : & if\ level\_of\_max(predict_{fitted}(OR_{add}, m)) = `\sim' \\ `-' : & if\ level\_of\_max(predict_{fitted}(OR_{add}, m)) = `-' \\ `--' : & otherwise \end{cases}$$

5 Analysis of Regression Models and Detection Strategies

In this section, we first describe and then analyze the ordinal regression models for our ADDs and then compare the detection strategies applied using these models.

5.1 Ordinal Regression Models

The final element required in our approach is the ordinal regression models for each of our ADDs. As explained, these are computed using R's *lrm* function from the *rms* package. As described in Section 3.5, the dependent outcome variables are the ground truth assessments for each ADD. The metrics defined in Table 3 are used as the independent predictor variables. The actual values, automatically computed with our detectors from the models of our data set, are reported in Table 4. The objective of the regression analysis is to predict the likelihood of the dependent outcome variable per ADDs.

In Table 5 we show the best three ordinal regression models we have found for each of the four ADDs. The p-value assesses the statistical significance of each regression model; the smaller the p-value, the stronger the model is. A p-value smaller than 0.05 is generally considered statistically significant. The C-index (which is also called the concordance index and is equivalent to the area under the Receiver Operating Characteristic (ROC) curve) is frequently reported in the statistical literature to measure the predictive power of ordinal regression models [17]. A C-index of 0.5 indicates random splitting, whereas a C-index of 1 indicates perfect prediction.

Harrel [13] suggests bootstrapping to obtain nearly unbiased estimates of a model's future performance based on re-sampling. A simple technique to adjust for optimism or overfitting is data splitting, but it is inefficient since the model is only fitted to a subset of the available data. Bootstrapping is thus the better and therefore a recommended method to adjust for optimism or overfitting [13]. We used lrm's *validate* function to perform bootstrapping and calculated the bias-corrected C-index in addition to the original C-index. The C-indexes, reported in Table 5, are all larger than 0.9. For each ADD, we have found at least two models with a bias-corrected C-index above or almost at 0.9, which indicates that the models are good enough for predicting the outcomes of individuals.

We used lrm's function *pentrace* to assist in the selection of penalty factors for fitting regression models using penalized maximum likelihood estimation (see [13]). In the reported models, we generally used a simple penalty of 1 and a non-linear penalty of 5. The penalized regression models offer slightly improved performance compared to non-penalized models.

Metric Description	Formal Definition
Authorized Backend Connectors (AUB). The $AUB : \mathbb{P}(CN_M) \rightarrow [0,1]$ metric applies the $authorized\_connectors$ detector to the distributed backend connectors requiring authorization. The number of successful detections is divided by the number of all these connectors. This determines the overall ratio of authorization in the backend.	$AUB(cn) = \frac{\lvert d\_success(authorized\_connectors(distributed\_backend\_connectors\_requiring\_authorization(cn)))\rvert}{\lvert distributed\_backend\_connectors\_requiring\_authorization(cn)\rvert}$
Backend Connectors Authorized with Authorization Tokens (AUB_T). $AUB\_T : \mathbb{P}(CN_M) \rightarrow [0,1]$ includes only the backend connectors with authorization based on the $TokenBasedAuthorization$ type (using the $authorized\_with\_authorization\_tokens$ detector).	$AUB\_T(cn) = \frac{\lvert d\_success(authorized\_with\_authorization\_tokens(distributed\_backend\_connectors\_requiring\_authorization(cn)))\rvert}{\lvert distributed\_backend\_connectors\_requiring\_authorization(cn)\rvert}$
Backend Connectors Authorized with Encrypted Information (AUB_E). $AUB\_E : \mathbb{P}(CN_M) \rightarrow [0,1]$ includes only the backend connectors with authorization based on the $EncryptedAuthorizationInformation$ type (using the $authorized\_with\_authorization\_tokens$ detector).	$AUB\_E(cn) = \frac{\lvert d\_success(authorized\_with\_encrypted\_information(distributed\_backend\_connectors\_requiring\_authorization(cn)))\rvert}{\lvert distributed\_backend\_connectors\_requiring\_authorization(cn)\rvert}$
Backend Connectors Authorized with Plaintext Information (AUB_P). $AUB\_P : \mathbb{P}(CN_M) \rightarrow [0,1]$ includes only the backend connectors with authorization based on the $AuthorizationWithPlaintextInformation$ type (using the $connectors\_authorized\_with\_plaintext\_information$ detector).	$AUB\_P(cn) = \frac{\lvert d\_success(connectors\_authorized\_with\_plaintext\_information(distributed\_backend\_connectors\_requiring\_authorization(cn)))\rvert}{\lvert distributed\_backend\_connectors\_requiring\_authorization(cn)\rvert}$
Authorized Backend Connectors Over a Secure Connection (AUB_C). The $AUB\_C : \mathbb{P}(CN_M) \rightarrow [0,1]$ metric calculates the ratio of the distributed backend connectors requiring authentication that offer a secure connection and are authorized in some manner to all these connectors.	$AUB\_C(cn) = \frac{\lvert d\_success(authorized\_connectors(d\_elements(d\_success(secure\_connectors(distributed\_backend\_connectors\_requiring\_authorization(cn))))))\rvert}{\lvert distributed\_backend\_connectors\_requiring\_authorization(cn)\rvert}$
Authorized Backend Connectors Using a Secure Method or Transferred Over a Secure Connection (AUB_A) The $AUB\_A : \mathbb{P}(CN_M) \rightarrow [0,1]$ metric calculates the ratio of those distributed backend connectors requiring authentication that either use authorization tokens, encrypted information, or authorization over a secure connection to all such connectors. $authorized\_connectors\_using\_secure\_method\_or\_communication : \mathbb{P}(CN_M) \rightarrow \mathbb{P}(DR)$ is a function returning all successful detector results for a set of connectors that use either authorization tokens, encrypted information, or a secure communication method.	$authorized\_using\_secure\_method\_or\_communication(cn) = d\_success(authorized\_with\_authorization\_tokens(cn)) \cup d\_success(authorized\_with\_encrypted\_information(cn)) \cup d\_success(authorized\_connectors(d\_elements(d\_success(secure\_connectors(cn)))))$ $AUB\_A(cn) = \frac{\lvert authorized\_using\_secure\_method\_or\_communication(distributed\_backend\_connectors\_requiring\_authorization(cn))\rvert}{\lvert distributed\_backend\_connectors\_requiring\_authorization(cn)\rvert}$
Authorized Connectors on Client/UI to Service Path (AUC). The $AUC : \mathbb{P}(CP_M) \rightarrow [0,1]$ metric applies the $authorized\_connectors$ detector to the client/service path connectors requiring authorization and divides the number of successful results by the number of all such connectors.	$AUC(cp) = \frac{\lvert d\_success(authorized\_connectors(client\_service\_path\_connectors\_requiring\_authorization(cp)))\rvert}{\lvert client\_service\_path\_connectors\_requiring\_authorization(cp)\rvert}$
Connectors on Client/UI to Service Path Authorized with Authorization Tokens (AUC_T). $AUC\_T : \mathbb{P}(CP_M) \rightarrow [0,1]$ includes only the client/service path connectors with authorization based on the $TokenBasedAuthorization$ type.	$AUC\_T(cp) = \frac{\lvert d\_success(authorized\_with\_authorization\_tokens(client\_service\_path\_connectors\_requiring\_authorization(cp)))\rvert}{\lvert client\_service\_path\_connectors\_requiring\_authorization(cp)\rvert}$
Connectors on Client/UI to Service Path Authorized with Encrypted Information (AUC_E). $AUC\_E : \mathbb{P}(CP_M) \rightarrow [0,1]$ includes only the client/service path connectors with authorization based on the $EncryptedAuthorizationInformation$ type.	$AUC\_E(cp) = \frac{\lvert d\_success(authorized\_with\_encrypted\_information(client\_service\_path\_connectors\_requiring\_authorization(cp)))\rvert}{\lvert client\_service\_path\_connectors\_requiring\_authorization(cp)\rvert}$
Connectors on Client/UI to Service Path Authorized with Plaintext Information (AUC_P). $AUC\_P : \mathbb{P}(CP_M) \rightarrow [0,1]$ includes only the client/service path connectors with authorization based on the $AuthorizationWithPlaintextInformation$ type.	$AUC\_P(cp) = \frac{\lvert d\_success(connectors\_authorized\_with\_plaintext\_information(client\_service\_path\_connectors\_requiring\_authorization(cp)))\rvert}{\lvert client\_service\_path\_connectors\_requiring\_authorization(cp)\rvert}$
Authorized Client/Service Path Connectors Over a Secure Connection (AUB_C). The $AUC\_C : \mathbb{P}(CP_M) \rightarrow [0,1]$ metric calculates the ratio of the client/service path connectors requiring authentication with a secure connection and some authorization to all such connectors.	$AUB\_C(cp) = \frac{\lvert d\_success(authorized\_connectors(d\_elements(d\_success(secure\_connectors(client\_service\_path\_connectors\_requiring\_authorization(cp))))))\rvert}{\lvert client\_service\_path\_connectors\_requiring\_authorization(cp)\rvert}$
Authorized Client/Service Path Connectors Using a Secure Method or Transferred Over a Secure Connection (AUC_A) The $AUC\_A : \mathbb{P}(CP_M) \rightarrow [0,1]$ metric calculates the ratio of the client/service path connectors requiring authentication that either use authorization tokens, encrypted information, or authorization over a secure connection to all such connectors.	$AUC\_A(cp) = \frac{\lvert d\_success(authorized\_using\_secure\_method\_or\_communication(client\_service\_path\_connectors\_requiring\_authorization(cp)))\rvert}{\lvert client\_service\_path\_connectors\_requiring\_authorization(cp)\rvert}$
Gateway Paths (GWP) The $GWP : \mathbb{P}(CP_M) \rightarrow [0,1]$ metric calculates the ratio of client/service paths containing a gateway to all such paths. The $gateway\_paths$ detector is successful for the paths that contain a component of type $API\_Gateway$ or $Backends\_for\_Frontsends$.	$GWP(cp) = \frac{\lvert d\_success(gateway\_paths(all\_paths\_from\_clients\_or\_uis\_to\_system\_services(cp)))\rvert}{\lvert all\_paths\_from\_clients\_or\_uis\_to\_system\_services(cp)\rvert}$
Frontend Service Paths (FEP) The $FEP : \mathbb{P}(CP_M) \rightarrow [0,1]$ metric calculates the ratio of client/service paths containing a frontend service to all such paths. The $frontend\_service\_paths$ detector is successful for the paths that contain a component that has the types $Facade$ and $Service$ and for which $gateway\_paths$ is not successful.	$FEP(cp) = \frac{\lvert d\_success(frontend\_service\_paths(all\_paths\_from\_clients\_or\_uis\_to\_system\_services(cp)))\rvert}{\lvert all\_paths\_from\_clients\_or\_uis\_to\_system\_services(cp)\rvert}$
Gateway or Frontend Service Paths (GFP) The $GFP : \mathbb{P}(CP_M) \rightarrow [0,1]$ metric calculates the ratio of client/service paths containing either a gateway or a frontend service to all such paths. The $gateway\_or\_frontend\_service\_paths$ detector is the union of results of the $frontend\_service\_paths$ and $gateway\_paths$ detectors.	$GFP(cp) = \frac{\lvert d\_success(gateway\_or\_frontend\_service\_paths(all\_paths\_from\_clients\_or\_uis\_to\_system\_services(cp)))\rvert}{\lvert all\_paths\_from\_clients\_or\_uis\_to\_system\_services(cp)\rvert}$
Components Without Plaintext Sensitive Data (CMP) The $CMP : \mathbb{P}(CP_M) \rightarrow [0,1]$ metric calculates the ratio of components containing no plaintext sensitive data (found with the $components\_without\_plaintext\_sensitive\_data$ detector) to all components.	$CMP(cp) = \frac{\lvert d\_success(components\_without\_plaintext\_sensitive\_data(cp))\rvert}{\lvert cp\rvert}$
Connectors Without Plaintext Sensitive Data (CNP) The $CNP : \mathbb{P}(CN_M) \rightarrow [0,1]$ metric calculates the ratio of connectors containing no plaintext sensitive data (found with the $connectors\_without\_plaintext\_sensitive\_data$ detector) to all connectors.	$CNP(cn) = \frac{\lvert d\_success(connectors\_without\_plaintext\_sensitive\_data(cn))\rvert}{\lvert cn\rvert}$
Components and Connectors Without Plaintext Sensitive Data (CCP) The $CCP : \mathbb{P}(ME_M) \rightarrow [0,1]$ metric calculates the ratio of components and connectors containing no plaintext sensitive data (found with the $components\_without\_plaintext\_sensitive\_data$ detector) to all components.	$CCP(me) = \frac{\lvert d\_success(components\_without\_plaintext\_sensitive\_data(components(me)) \cup connectors\_without\_plaintext\_sensitive\_data(connectors(me)))\rvert}{\lvert components(me) \cup connectors(me)\rvert}$

TABLE 3: Metric Definitions

Metric	AC0	AC1	AC2	BA0	BA1	BA2	CI0	CI1	CI2	CO0	CO1	CO2	EP0	EP1	EP2	ES0	ES1	ES2	OB0	OB1	OB2	PM0	PM1	PM2	RS0	RS1	RS2	TE0	TE1	TE2
AUB	0.22	0.88	1.0		0.93	0.81		0.91	1.0		0.5	1.0		1.0	1.0	0.17	0.83	1.0		0.94	1.0	0.14	1.0	1.0		0.73	1.0		1.0	1.0
AUB_T	0.22		1.0			0.25								0.33	1.0	0.17	0.26	0.24				0.14								
AUB_E	0.22		1.0			0.25		0.36	0.31			0.38		0.67	1.0	0.17	0.61	0.56		0.78	0.32	0.14				0.73	0.64			0.08
AUB_P		0.88			0.93	0.56		0.55	0.69		0.5	0.62		0.33			0.22	0.44		0.17	0.68		1.0	1.0			0.36		1.0	0.92
AUB_C		0.38	1.0		0.53	0.81		0.55	0.69			1.0		0.33	1.0		0.22	1.0		0.06	1.0		0.61	1.0		0.23	0.4		0.17	0.24
AUB_A	0.22	0.38	1.0		0.53	0.81		0.91	1.0			1.0		1.0	1.0	0.17	0.83	1.0		0.78	1.0	0.14	0.61	1.0		0.73	1.0		0.17	0.32
AUC	1.0	0.5	1.0		1.0	0.83		0.86	1.0		1.0	1.0		1.0	1.0	0.4	0.33	1.0		0.93	1.0	1.0	1.0	1.0		0.83	1.0		1.0	0.67
AUC_T	1.0		1.0			0.83								0.5	1.0	0.4	0.33	0.71				1.0								
AUC_E	1.0		1.0			0.83								1.0	1.0	0.4	0.33	0.71		0.43	0.67	1.0				0.83				0.33
AUC_P		0.5			1.0			0.86	1.0		1.0	1.0						0.29		0.5	0.33		1.0	1.0			1.0		1.0	0.33
AUC_C			1.0			0.83		0.86	1.0		0.7	1.0			1.0			1.0			1.0		0.38	1.0		0.42	1.0			
AUC_A	1.0		1.0			0.83		0.86	1.0		0.7	1.0		1.0	1.0	0.4	0.33	1.0		0.43	1.0	1.0	0.38	1.0		0.83	1.0			0.33
GWP	1.0		0.5	1.0	0.8	0.83	1.0	0.91	0.85				1.0	0.38	0.33	1.0	0.65	0.71				1.0	0.5	0.67	1.0	0.92	0.67			
FEP								0.09	0.15		0.15	0.7		0.5	0.67				1.0	0.49	0.5		0.5	0.33						
GFP	1.0		0.5	1.0	0.8	0.83	1.0	1.0	1.0		0.15	0.7	1.0	0.88	1.0	1.0	0.65	0.71	1.0	0.49	0.5	1.0	1.0	1.0	1.0	0.92	0.67			
CMP	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.92	0.83	0.62	0.58	0.58	1.0	0.81	1.0	1.0	1.0	1.0	1.0	0.77	0.92	0.88	0.89	0.88	1.0	0.95	0.74	1.0	1.0	1.0
CNP	0.5	0.27	0.86	0.18	0.33	0.38	0.67	0.71	0.78	0.75	0.84	0.81	0.73	0.91	0.87	0.76	0.81	0.8	1.0	1.0	1.0	0.62	0.65	0.59	1.0	1.0	1.0	1.0	1.0	1.0
CCP	0.71	0.58	0.91	0.5	0.58	0.58	0.8	0.79	0.8	0.69	0.74	0.71	0.86	0.87	0.92	0.85	0.87	0.87	1.0	0.92	0.98	0.7	0.72	0.69	1.0	0.98	0.91	1.0	1.0	1.0

TABLE 4: Metric Calculation Results for the Case Study Systems

The reported models in Table 5 do not always use the complete set of our metrics. As recommended, we applied data reduction [13] (i.e., eliminating variables from the models to find models that performed. We only report those three with the highest bias-corrected C-indexes among the numerous models that we tested. As a consequence, all suggested metrics are relevant in our prediction models, but in no decision all of them are needed.

5.2 Detection Strategy Analysis

To analyze and compare the use of our models in the two alternative detection strategies defined in Section 4.6, we calculate the Mean Square Error (MSE) and Spearman correlation. MSE is commonly used as an evaluation measure for ordinal regression predictions [18], [19]. Spearman correlation can be additionally utilized for error-sensitive evaluation for ordinal target variables [19].

The results are reported in Table 6. First of all, it can be seen that all strategy/model combinations have very high positive correlation values for all ADDs (all > 0.9). The MSE values overall mostly show low errors in the prediction. The direct comparison of the values, together with the bootstrapped C-Index values from Table 5, enable us to select the best strategy/model combinations for our data set per ADD:

- For BE_AU, all strategy/model combinations have low MSE values. Overall, the Means Predictor Detection Strategy combined with Model 3 offers the best performance in our data set, which also provides a very high bootstrapped C-Index value.
- For CP_AU, all strategy/model combinations, the Fitted Predictor Detection Strategy combined with Models 1 or 2 offers the best MSE values. While Model 2 has a slightly better MSE value, Model 1 has a slightly better bootstrapped C-Index value.
- For TC, only one MSE value < 0.25 can be observed in the Fitted Predictor Detection Strategy of Model 3. But this model has the worst bootstrapped C-Index of 0.76. This indicates slight overfitting in this model. Thus, combining the Means Predictor Detection Strategy with Model 1, offering an MSE value of 0.25, a very high bootstrapped C-Index, and a very high correlation seems to be the strategy with the best performance.
- For SD, both strategies perform very well with Model 3. As this has a bootstrapped C-Index of 0.9, selecting one of them is a good option. If a bootstrapped C-Index of > 0.9 is needed, the Means Predictor Detection Strategy with Model 2 performs best.

6 Automated Application of Detection Strategies for Resilience Assessment

To further evaluate our approach, we have applied it to an industrial resilience assessment tool currently being developed by EU-VRi. Resilience assessment refers to evaluating a system's ability to withstand and recover from adverse conditions or attacks. Our detection strategies approach can be utilized in the resilience assessment of a system as an automated component (once the architecture model extraction has been specified, see Section 2). The tool aims to assess, monitor and optimize the resilience of a system based on existing industry guidelines such as the ones studied in our data collection and analysis step (described in Section 2).

The tool provides an API to integrate building blocks, such as the metrics provided by our approach. It provides an indicator-based approach to calculate a resilience level index for a system as a composite, multi-level indicator. The tool supports before/after analysis, multi-assessment monitoring over time, and decision support based on sensitivity analysis. As we expect the tool to perform automated assessments, we have designed our metrics and models to be integrated within this tool as an automated component.

Our lessons learned are that the approach can be integrated into existing tools and processes for resilience assessment. By integrating the approach into a resilience assessment tool, developers, architects, or assessors can automatically calculate metrics for the software architectural parts of a composite resilience level index. This can assist in identifying areas where improvements can be made and in evaluating the overall resilience of a system.

ADD	Measure Name	Model 1	Model 2	Model 3
Backend Authorization (BE_AU)	*Metrics used in the model*	AUB_T, AUB_P, AUB_A	AUB, AUB_P, AUB_C, AUB_A	AUB_T, AUB_E, AUB_P, AUB_C
	Model p-value	3.883560e-13	2.167155e-13	1.191269e-13
	C-Index (original)	0.9699140	0.9885387	0.9828080
	C-Index (bootstrapped, bias-corrected)	0.9548209	0.916884	0.9554083
Authorization on Paths from Clients or UIs to System Services (CP_AU)	*Metrics used in the model*	AUC_T, AUC_E, AUC_P, AUC_C	AUC, AUC_P, AUC_C	AUC_E, AUC_P, AUC_C, AUC_A
	Model p-value	2.971512e-12	8.518408e-12	5.328848e-12
	C-Index (original)	0.9719101	0.9382022	0.9719101
	C-Index (bootstrapped, bias-corrected)	0.9639256	0.9204635	0.938125
Traffic Control (TC)	*Metrics used in the model*	GWP, GFP	GWP, FEP	FEP, GFP
	Model p-value	5.780931e-13	2.275191e-11	2.157718e-12
	C-Index (original)	0.9805014	0.9832869	0.9916435
	C-Index (bootstrapped, bias-corrected)	0.9484889	0.9343593	0.7607799
Avoiding Plaintext Sensitive Data (SD)	*Metrics used in the model*	CMP, CNP	CNP, CCP	CMP, CNP, CCP
	Model p-value	2.285205e-11	6.307177e-13	9.261480e-13
	C-Index (original)	0.9907121	0.9907121	0.9969040
	C-Index (bootstrapped, bias-corrected)	0.9141409	0.9402864	0.8958514

TABLE 5: Regression Analysis Results

ADD	Detection Strategy	Measure Name	Model 1	Model 2	Model 3
Backend Authorization (BE_AU)	*Means Predictor Detection Strategy*	*Mean Squared Error (MSE)*	0.1557119	0.1468277	0.1277475
		Spearman Correlation	0.9469793	0.9672109	0.9600839
	Fitted Predictor Detection Strategy	*Mean Squared Error (MSE)*	0.2	0.1666667	0.1666667
		Spearman Correlation	0.9309631	0.9585881	0.9380393
Authorization on Paths from Clients or UIs to System Services (CP_AU)	*Means Predictor Detection Strategy*	*Mean Squared Error (MSE)*	0.1834024	0.2569814	0.2160254
		Spearman Correlation	0.9495478	0.9198208	0.9592745
	Fitted Predictor Detection Strategy	*Mean Squared Error (MSE)*	0.1666667	0.1333333	0.3
		Spearman Correlation	0.9583582	0.9666581	0.92638
Traffic Control (TC)	*Means Predictor Detection Strategy*	*Mean Squared Error (MSE)*	0.2529702	0.3094919	0.2644267
		Spearman Correlation	0.9688093	0.9715577	0.9798029
	Fitted Predictor Detection Strategy	*Mean Squared Error (MSE)*	0.2666667	0.3	0.2333333
		Spearman Correlation	0.939046	0.934118	0.9537935
Avoiding Plaintext Sensitive Data (SD)	*Means Predictor Detection Strategy*	*Mean Squared Error (MSE)*	0.1478888	0.1267839	0.1128048
		Spearman Correlation	0.94364	0.94364	0.9525137
	Fitted Predictor Detection Strategy	*Mean Squared Error (MSE)*	0.1666667	0.1666667	0.1
		Spearman Correlation	0.9084704	0.9223994	0.9492435

TABLE 6: Comparison of Detection Strategies

The approach is scalable and can be applied to systems of varying sizes. As microservice-based systems are highly modular, each service can be analyzed separately for the design options of the ADDs. Thus, in our experience, the scale of the system under investigation plays only a minor role, and the analysis results also tend to be well-fitting for larger applications. However, the application's scale does play a role in understanding the automated assessment results. For large-scale systems, the metrics generated by the approach are better suited for understanding the assessment results, while for small-scale systems, inspecting the models can be enough. In general, the metrics generated by the approach are easy to understand and can help identify areas for improvement. The models are better suited for inspecting specific service issues or issues of a few interacting services.

The approach has a limitation in that it relies on ADDs to model security recommendations, which may not encompass all pertinent security concerns. However, in the resilience assessment tool, these are supplied by other components. Furthermore, our approach mandates manual model creation or calibration for constructing a regression model. Therefore, if additional systems with distinct practices are not well reflected in our open-source systems dataset, they must be manually included before our approach can be employed for such cases. Nevertheless, microservice systems' modularity permits adding a few mid-sized systems (e.g., open-source) to our data set to incorporate these practices. Thus, there is no need to manually analyze large-scale systems to apply our approach at scale, which is a positive aspect.

7 Discussion and Threats to Validity

In this section, we first discuss our lessons learned and then discuss potential threats to validity.

7.1 Discussion of Research Questions

In **RQ1**, we have aimed to investigate how to detect conformance to ADDs for microservice system security automatically. Our proposal is based on an analysis of experts' security recommendations, modeling them as ADDs and then letting the experts judge these recommendations using an ordinal scheme typically used in such human assessments. All further steps can then be automated based on a suitable model data set, like ours. Our open-source model data set is thus also a major contribution of this article, as such datasets are needed to calibrate statistical models like ours.

For this first step of our approach, this article has essentially provided validation and extension of the methods introduced in our prior work [11]. Based on a different set of ADD and metrics and a substantially extended formalization, we again achieved excellent regression results, as

reported in Section 5.1. Thus, we are confident that these models are a good basis for our new approach and core contribution, the detection strategies, analyzed in Section 5.2.

Overall, we have proposed 24 detection strategies (two kinds of strategies applied for each of the three regression models found for each of the four ADDs). The novel detection strategies are metrics-based rules. In contrast to earlier detection strategy proposals [10], our approach is based on a statistical model's predictions to avoid the problem of having to decide on metric value thresholds for the rules manually. The original detection strategy proposal [10] used simple data filters like $HigherThan(value)$ or $Between(value1, value2)$, which inherently leads to the issue of how to select appropriate threshold values for the rules. In contrast, our novel approach uses two kinds of statistical predictors and tests each for multiple models to develop the best-performing models.

The comparison of the 24 detection strategies then provided the answer to **RQ2**. In particular, our results show that all 24 detection strategies show a very high Spearman correlation (> 0.9) to the ground truth of our model data set. Overall, all 24 reported strategy/model combinations perform reasonably well. In Table 5, we only report the three best regression models found. Many models were tested that did not perform well or showed significant bias (e.g., due to optimism or overfitting). Essentially by data reduction (eliminating variables from the models), we found models that performed well in all considered measures. Using detailed analysis, we further were able in Section 5 to select the few best performing models: For all but the ADD TC, we found at least one strategy with an MSE value < 0.2, for most, even < 0.15. The lowest value for TC is also low (0.23). In the regression analysis, we used bootstrapping as the recommended technique to avoid bias due to optimism or overfitting [13], and for each ADD one detection strategy with a low MSE value, a very high Spearman correlation (> 0.9), and a very low bootstrapped C-index (< 0.9) was found.

Our approach uses the paths described in Section 4.3. This was used for analyzing specific propagations of security flaws in a system, e.g., if a service can be transitively reached from a client without proper authorization measures. The paths enable many other analysis options regarding the potential propagation of security flaws in a system. For example, in our prior works, we have developed a method for avoiding excessive data exposure in microservice APIs [20]. Our paths would enable checking for this flaw in the whole microservice backend. Many other such analyses are possible for future work.

As shown by our analysis of 10 mid-size open-source systems created by practitioners and our experience in applying our approach in the context of the resilience assessment tool, summarized in Section 6, we can assess that the proposed approach can be used in practice to assess architectural security and resilience aspects of a system. As discussed in Section 6, our experience shows that applying our approach as an automated component for systems using similar techniques as those used in the open-source systems dataset for creating the regression model is possible. Otherwise, extending the dataset with additional systems is necessary to reflect the missing techniques (which can be done as an extension to our dataset provided in our replication package). If other ADDs are to be analyzed than the ones modeled in our approach, manual dataset creation, annotation, and initial analysis are needed. If the systems to be analyzed follow a modular distributed systems approach, such as with microservices, our experience is that it is possible to scale the approach to larger systems based on a dataset of mid-sized systems. This is positive as it reduces the upfront manual work required. For this reason, we have limited our study to microservice systems.

While we do not claim that our approach can detect something not identifiable with other techniques, the use of ADDs and detection strategies provides a unique way to assess architectural security concerns in microservice-based systems. The analysis of the introduced ADDs can require substantial effort even for mid-size systems in our dataset (e.g., inspecting each direct or transitive client-service path) and is required after each system modification. Our detection strategies can be run as part of a more extensive automated analysis (e.g., as discussed in Section 6) or as an automated check in a continuous delivery pipeline. This way, e.g., accidentally introduced security issues can be spotted during a resilience assessment or before a system gets deployed without repeated manual effort."

7.2 Threats to Validity

We mainly relied on third-party systems as the basis for our study to increase internal validity and thus avoid bias in system composition and structure. It is possible that our search procedures resulted in some unconscious exclusion of specific sources; we mitigated this by assembling a team of authors with many years of experience in the field (including three industry experts) and conducting a very general and broad search. Because our search was not exhaustive and practitioners created the systems we found for demonstration purposes, i.e., they were relatively modest in size. This means that some potential architectural elements were not included in our metamodel. Furthermore, this poses a potential threat to the external validity of generalization to other, more complex systems. However, we are confident that the documented systems are a representative cross-section of current practice in this area. Another potential risk is the fact that the author team developed the system variants. However, we did this following best practices documented in the literature and with reviews from industrial experts. We were careful to change only certain aspects in a variant and keep all other elements stable.

Another possible source of internal validity impairment is the modeling process. The author team has considerable experience with similar methods, and the systems' models have been repeatedly and independently cross-checked, but the possibility of some interpretive bias remains. Other researchers may have coded or modeled differently, resulting in different models. Because our goal was only to find a model that could describe all observed phenomena, and we achieved this, we do not consider this risk to be particularly problematic for our study. The individual metrics used to assess the presence of each pattern were deliberately kept as simple as possible to avoid false positives and allow for a technology-independent assessment.

However, it might be the case that the expert judgment for the ground truth would differ for substantially different kinds of systems, e.g., systems from other domains or substantially larger systems. Then it would be necessary to re-run our statistical analysis with data from a few such systems to calibrate the prediction models to the changed circumstances. Thanks to the complete automation of our approach, once suitable models are created (modeled or reconstructed), the analysis steps can be automated.

To avoid threats concerning the generalizability of our approach, we limited our scope to microservice-based systems, even though some aspects of our approach are likely applicable to other kinds of distributed systems than microservice-based systems.

We do not claim completeness of the detection strategies or the metrics we present in this article. They are only complete in the sense that they cover all options of the ADDs they address.

8 Related Work

This section compares related works on tactics, best practices and patterns, detection strategies, and conformance checking in general, and then specialized metrics-based approaches for security and microservices.

8.1 Related Works on Tactics, Best Practices, and Patterns

The collection and systematization of microservice patterns have been the subject of much research. Richardson [3] collected microservice patterns related to key design and architectural practices. Zimmermann et al. [21] presented microservice API-related patterns. Skowronski [22] collected best practices for event-driven microservice architectures. Microservice fundamentals and best practices are also discussed by Fowler and Lewis [2] and summarized in a mapping study by Pahl and Jamshidi [23]. Taibi and Lenarduzzi [24] examined microservice bad smells, i.e., practices that developers should avoid, which in our work would correspond to ADD violations. This paper uses such guidance as a basis for modeling microservice architectures.

Similarly, attempts have been made to define security patterns [25], [26]. Microservice-specific recommendations from industry organizations [5], [6], [7] are proposed that represent broad-level summaries of existing industry best practices. We used such guidelines to guide our selection of security practices for study in our work.

8.2 Related Works on Detection Strategies and Conformance Checking

Detection strategies [10] are metrics-based rules for detecting design flaws. Whereas our work focuses on architectural design flaws for security and microservice best practices, the original detection strategies approach by Marinescu addresses generic object-oriented design smells. Whereas our approach leverages statistical predictors in constructing the metric-based rules, Marinescu uses simple data filters such as $HigherThan(value)$ or $Between(value1, value2)$. Thus, in such approaches, the issue of defining threshold values for parameterizing the detection strategies can introduce substantial bias. In our approach, this can be fully automated, and developers can measure the accuracy of the result with established measures such as the bootstrapped C-index, MSE, and Spearman correlation. The following two sections discuss specialized detection approaches based on security and microservice metrics.

Conformance assessment has been applied in various areas of software engineering, such as service composition [8] and traceability to guidelines [27]. In general, the conformance relation is defined as the consistency between models [8]. In software architecture, conformance assessment addresses the relation between a software system's architecture and its intended architecture [9]. Our approach shares with those works the general notion of architectural conformance assessment.

Once metrics can be checked automatically, our approach can be classified as a metrics-based, microservice-specific approach for software architecture conformance checking. In general, approaches for architecture conformance checking are often based on automated extraction techniques [28], [29]. Conformance to architecture patterns [28], [30] or other architectural rules [29] can usually be checked by such approaches. Techniques based on a broad set of microservice-related metrics to cover multiple microservice tenets and security do not yet exist.

8.3 Related Works on Security Metrics-Based Approaches

Security experts can use security metrics to understand the current security state and potentially improve it [31]. While several organizations such as Microsoft [32] and OWASP [33] propose processes and checklists for building secure architectures, very few tools can automate these processes for tailored solutions [34] due to the dynamic and polyglot nature of these systems.

Ramos et al. [35] conducted a detailed review of the main existing model-based quantitative security metrics focusing on network security metrics. Model-based security metrics use techniques to describe a system using an abstract model that captures necessary attributes based on attacker assumptions and system behavior. Noel et al. [36] describe a set of metrics for measuring network-wide cybersecurity risk based on a vulnerability model to multi-stage attacks. Their system for calculating security metrics from vulnerability-based network attack graphs uses data imported from sources commonly used in enterprise networks, such as vulnerability scanners and firewall configuration files. Attack Graphs are a model widely used to quantify network security. It uses the causal relationships between vulnerabilities, quantifying the likelihood of potential multi-step attacks that combine multiple vulnerabilities [37].

Such general security metrics and indicators are a foundation for our work. However, since none of them considers the specifics of microservice architectures, they cannot be applied to our research problems or only with significant adaptations. For this reason, we decided to develop metrics based on existing recommendations (such as NIST [5], OWASP [6], or Cloud Security Alliance [7]) specifically for microservice-based systems.

8.4 Related Works on Microservice Metrics-Based Approaches

Several studies have used metrics to assess microservice-based software architectures. Pautasso and Wilde [38] propose a composite, facet-based metric for assessing loose coupling in service-oriented systems. Zdun et al. [16] study the independent deployment of microservices based on metrics for evaluating architecture conformance to microservice patterns. Bogner et al. [39] propose a maintainability quality model which combines eleven easily extracted code metrics into a broader quality assessment. Engel et al. [40] present a method using real-time system communication traces to compute metrics on conformance to recommended microservice design principles such as loose coupling and small service size. Each of these approaches is focused on narrow sets of architecture-relevant tenets (e.g., loose coupling). Still, no general approach for an assessment across different aspects, such as the security-related ADDs in our work, exists.

While a couple of works specifically focus on microservice security metrics, many of these focus on runtime-related or non-architectural aspects [41], [42], [43], [44]. Chondamrongkul et al. [45] present an early approach to automatically investigate specific security vulnerabilities in a decomposition architecture, such as man-in-the-middle or denial-of-service attacks. In contrast, our work is based on ADD models, suggests detection strategies, and is based on empirical data, whereas the work of Chondamrongkul et al. uses only modeling examples.

9 Conclusion

In this article, a novel approach for automated conformance checking of ADD-based security recommendations for polyglot microservice systems based on detection strategies was developed. The detection strategies and novel metrics for microservice security were formally defined and implemented in a model-based tool. They were applied to an open source model dataset for security in software decomposition models of 10 open source systems and 20 variants of these systems (modeling possible conformance violations and/or improvements to test the suitability of our method for continuous evolution) and in an industrial resilience assessment tool. The novel detection strategies are metrics-based rules. Unlike previous proposals for detection strategies [10], our approach is based on the predictions of statistical models to avoid the problem of having to set thresholds for metric values for the rules manually. We proposed a total of 24 detection strategies (two types of strategies for each of the three regression models found for each of the four ADDs). The strategies and their metrics were statistically evaluated and compared. We found at least one strategy per ADD with a very high correlation and a low MSE value that also had a very low estimated bias (e.g., due to overfitting) according to the bootstrapped C-index value.

In the future, we plan to develop detection strategies for metrics that we have developed in previous work and to develop novel traceability strategies that allow root cause analysis based on detection strategies.

10 Acknowledgments

Our work has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 952647 (AssureMOSS project). This work was supported by: FWF (Austrian Science Fund) project API-ACE: I 4268; FWF (Austrian Science Fund) project IAC^2: I 4731-N. We thank the two experts, Gergely Eberhardt and Ákos Milánkovich from SEARCH-LAB, for reviewing our models, metrics, and code.

References

[1] S. Newman, *Building Microservices: Designing Fine-Grained Systems*. Sebastopol, CA: O'Reilly, 2015.

[2] J. Lewis and M. Fowler, "Microservices: a definition of this new architectural term," http://martinfowler.com/articles/microservices.html, Mar. 2004.

[3] C. Richardson, "A pattern language for microservices," http://microservices.io/patterns/index.html, 2017.

[4] O. Zimmermann, "Microservices tenets," *Computer Science-Research and Development*, vol. 32, no. 3-4, pp. 301–310, Jul. 2017.

[5] NIST, "Nist special publication (sp) 800-204, security strategies for microservices-based application systems," https://www.nist.gov/news-events/news/2019/08/security-strategies-microservices-based-application-systems-nist-publishes, 2019.

[6] OWASP, "Microservices based security arch doc cheat sheet," https://cheatsheetseries.owasp.org/cheatsheets/Microservices_based_Security_Arch_Doc_Cheat_Sheet.html, 2021.

[7] Cloud Security Alliance, "Best practices in implementing a secure microservices architecture," https://cloudsecurityalliance.org/artifacts/best-practices-in-implementing-a-secure-microservices-architecture/, 2020.

[8] D. Quartel and M. van Sinderen, "On interoperability and conformance assessment in service composition," in *11th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2007)*. IEEE, 2007, pp. 229–229.

[9] F. Deissenboeck, L. Heinemann, B. Hummel, and E. Juergens, "Flexible architecture conformance assessment with conqat," in *2010 ACM/IEEE 32nd International Conference on Software Engineering*, vol. 2. Washington, DC, USA: IEEE, 2010, pp. 247–250.

[10] R. Marinescu, "Detection strategies: Metrics-based rules for detecting design flaws," in *20th IEEE International Conference on Software Maintenance, 2004. Proceedings.* IEEE, 2004, pp. 350–359.

[11] U. Zdun, P.-J. Queval, G. Simhandl, R. Scandariato, S. Chakravarty, M. Jelic, and A. Jovanovic, "Microservice security metrics for secure communication, identity management, and observability," Transactions on Software Engineering and Methodology (TOSEM), ACM, accepted for publication, online accessible at: https://dl.acm.org/doi/10.1145/3532183, 2023.

[12] E. Ntentos, U. Zdun, K. Plakidas, P. Genfer, S. Geiger, S. Meixner, and W. Hasselbring, "Detector-based component model abstraction for microservice-based systems," *Computing*, vol. 103, p. 2521–2551, 2021.

[13] J. Frank E. Harrell, *Regression Modeling Strategies: With Applications to Linear Models, Logistic and Ordinal Regression, and Survival Analysis*, 2nd ed. Berlin, Heidelberg: Springer, 2015.

[14] F. E. Harrell, "Package 'rms'," https://cran.r-project.org/web/packages/rms/rms.pdf, 2022.

[15] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[16] U. Zdun, E. Navarro, and F. Leymann, "Ensuring and assessing architecture conformance to microservice decomposition patterns," in *Service-Oriented Computing*, M. Maximilien, A. Vallecillo, J. Wang, and M. Oriol, Eds. Cham: Springer International Publishing, 2017, pp. 411–429.

[17] A. Airola, T. Pahikkala, W. Waegeman, B. De Baets, and T. Salakoski, "An experimental comparison of cross-validation techniques for estimating the area under the roc curve," *Computational Statistics & Data Analysis*, vol. 55, no. 4, pp. 1828–1844, 2011.

[18] S. Baccianella, A. Esuli, and F. Sebastiani, "Evaluation measures for ordinal regression," in *2009 Ninth International Conference on Intelligent Systems Design and Applications*, 2009, pp. 283–287.

[19] D. Chen, M. Courtland, A. Faulkner, and A. Ezen-Can, "Error-sensitive evaluation for ordinal target variables," in *Proceedings of the 2nd Workshop on Evaluation and Comparison of NLP Systems*. Punta Cana, Dominican Republic: Association for Computational Linguistics, Nov. 2021, pp. 189–199.

[20] P. Genfer and U. Zdun, "Avoiding excessive data exposure through microservice apis," in *Software Architecture*, I. Gerostathopoulos, G. Lewis, T. Batista, and T. Bureš, Eds. Cham: Springer International Publishing, 2022, pp. 3–18.

[21] O. Zimmermann, M. Stocker, U. Zdun, D. Luebke, and C. Pautasso, "Microservice API patterns," https://microservice-api-patterns.org, 2019.

[22] J. Skowronski, "Best practices for event-driven microservice architecture," https://hackernoon.com/best-practices-for-event-driven-microservice-architecture-e034p21lk, 2019.

[23] C. Pahl and P. Jamshidi, "Microservices: A systematic mapping study," in *6th International Conference on Cloud Computing and Services Science*. Setubal, PRT: SCITEPRESS, 2016, pp. 137–146.

[24] D. Taibi and V. Lenarduzzi, "On the definition of microservice bad smells," *IEEE Software*, vol. 35, no. 3, pp. 56–62, May 2018.

[25] M. Hafiz, P. Adamczyk, and R. E. Johnson, "Growing a pattern language (for security)," in *Proceedings of the ACM international symposium on New ideas, new paradigms, and reflections on programming and software*. New York, NY, USA: Association for Computing Machinery, 2012, pp. 139–158.

[26] M. Schumacher, E. Fernandez-Buglioni, D. Hybertson, F. Buschmann, and P. Sommerlad, *Security Patterns: Integrating security and systems engineering*. New York, NY, USA: John Wiley & Sons, 2013.

[27] P. Rempel, P. Mäder, T. Kuschke, and J. Cleland-Huang, "Mind the gap: assessing the conformance of software traceability to relevant guidelines," in *Proceedings of the 36th International Conference on Software Engineering*, 2014, pp. 943–954.

[28] G. Y. Guo, J. M. Atlee, and R. Kazman, "A software architecture reconstruction method," in *Software Architecture*. Berlin, Heidelberg: Springer, 1999, pp. 15–33.

[29] A. Van Deursen, C. Hofmeister, R. Koschke, L. Moonen, and C. Riva, "Symphony: View-driven software architecture reconstruction," in *4th Working IEEE/IFIP Conf. on Software Architecturen(WICSA 2004)*, IEEE. Washington, DC, USA: IEEE, 2004, pp. 122–132.

[30] T. Haitzer and U. Zdun, "Semi-automated architectural abstraction specifications for supporting software evolution," *Science of Computer Programming*, vol. 90, pp. 135–160, 2014.

[31] S. C. Payne, "A guide to security metrics," SANS Institute Information Security Reading Room, 2006.

[32] J. Ingeno, *Software Architect's Handbook: Become a successful software architect by implementing effective architecture concepts*. Birmingham: Packt Publishing Ltd, 2018.

[33] M. Woschek, "Owasp cheat sheets," *pp*, vol. 315, p. 4, 2015.

[34] P. Parrend, T. Mazzucotelli, and F. Colin, "Using design structure matrices (dsm) as security controls for software architectures," Tech. Rep. 1, Complex System Digital Campus, cS-DC Research Report, ARK . . . , Tech. Rep., 2017.

[35] A. Ramos, M. Lazar, R. Holanda Filho, and J. J. Rodrigues, "Model-based quantitative network security metrics: A survey," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2704–2734, 2017.

[36] S. Noel and S. Jajodia, "A suite of metrics for network attack graph analytics," in *Network security metrics*. Berlin, Heidelberg: Springer, 2017, pp. 141–176.

[37] L. Wang, T. Islam, T. Long, A. Singhal, and S. Jajodia, "An attack graph-based probabilistic security metric," in *Data and Applications Security XXII*, V. Atluri, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 283–296.

[38] C. Pautasso and E. Wilde, "Why is the web loosely coupled?: a multi-faceted metric for service design," in *18th Int. Conf. on World wide web*. New York, NY, USA: Association for Computing Machinery, 2009, pp. 911–920.

[39] J. Bogner, S. Wagner, and A. Zimmermann, "Towards a practical maintainability quality model for service-and microservice-based systems," in *Proceedings of the 11th European Conference on Software Architecture: Companion Proceedings*, ser. ECSA '17. New York, NY, USA: ACM, 2017, p. 195–198.

[40] T. Engel, M. Langermeier, B. Bauer, and A. Hofmann, "Evaluation of microservice architectures: A metric and tool-based approach," in *Information Systems in the Big Data Era*, J. Mendling and H. Mouratidis, Eds. Cham: Springer International Publishing, 2018, pp. 74–89.

[41] K. A. Torkura, M. I. Sukmana, A. V. Kayem, F. Cheng, and C. Meinel, "A cyber risk based moving target defense mechanism for microservice architectures," in *16th IEEE International Symposium on Parallel and Distributed Processing with Applications (ISPA 2018)*. Washington, DC, USA: IEEE, 2018, pp. 932–939.

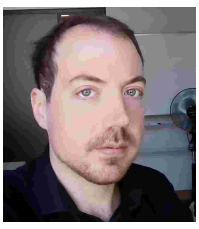
[42] A. Avritzer, "Challenges and approaches for the assessment of micro-service architecture deployment alternatives in devops: A tutorial presented at icsa 2020," in *2020 IEEE International Conference on Software Architecture Companion (ICSA-C)*. Washington, DC, USA: IEEE, 2020, pp. 1–2.

[43] J. Levin and T. A. Benson, "Viperprobe: Rethinking microservice observability with ebpf," in *2020 IEEE 9th International Conference on Cloud Networking (CloudNet)*. Washington, DC, USA: IEEE, 2020, pp. 1–8.

[44] J. Flora, "Improving the security of microservice systems by detecting and tolerating intrusions," in *2020 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. Washington, DC, USA: IEEE, 2020, pp. 131–134.

[45] N. Chondamrongkul, J. Sun, and I. Warren, "Automated security analysis for microservice architecture," in *2020 IEEE International Conference on Software Architecture Companion (ICSA-C)*. Washington, DC, USA: IEEE, 2020, pp. 79–82.

Uwe Zdun is a full professor for software architecture at the Faculty of Computer Science, University of Vienna. His research focuses on software design and architecture, distributed systems engineering (service-based, cloud, IoT, and microservices systems), DevOps and continuous delivery, and empirical software engineering. Uwe has published more than 275 peer-reviewed articles and is co-author of 4 professional books: "Patterns for API Design: Simplifying Integration with Loosely Coupled Message Exchanges", "Remoting Patterns - Foundations of Enterprise, Internet, and Realtime Distributed Object Middleware", "Process-Driven SOA - Proven Patterns for Business-IT Alignment", and "Software-Architektur." He is Associate Editor of the Journal of Systems and Software (JSS) published by Elsevier, Associate Editor of the Computing journal published by Springer, and Associate Editor-in-Chief for design and architecture for the IEEE Software magazine.



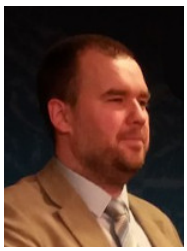
Pierre-Jean Quéval is PhD student and scientific project staff in the "Software Architecture" research group of the University of Vienna. His current research interests are the automation of software architecture recovery and modeling, and the use of heuristics and inference in general.

Georg Simhandl Georg Simhandl is a senior researcher at the Faculty of Computer Science, University of Vienna. Before joining the Research Group Software Architecture, he founded Adaptivia GmbH (in 2008), a company focusing on Intelligent Sensor Networks where he led numerous R&D projects. Before that, Georg was a software architect, consultant and product manager in big consulting, industrial and startup companies. Georg holds a PhD in Information Systems Research (2007) from the Vienna University of Business Administration and Economics. His research focuses on software architecture at the crossroad of Distributed Systems, Cyber Security and Cognitive Computing.

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2023.3276487

Riccardo Scandariato Prof. Dr. Riccardo Scandariato received his PhD in Computer Science in 2004 from Politecnico di Torino, Italy. He is a full professor at the Hamburg University of Technology (TUHH), in Germany, where he leads the Institute of Software Security. Together with his team, he applies an interdisciplinary approach to create innovative tools and techniques for designing and implementing secure, privacy-friendly applications. His core research topics are model-based security, prediction of software vulnerabilities, automatic vulnerability repair, and usable security.

Somik Chakravarty (PhD) holds a doctorate from Sorbonne Universités - UTC in Process Engineering. He received an EU Marie Curie ITN fellowship for completing his doctorate in collaboration with the INERIS, France. He was awarded the Jean Bricard Prix and nominated for the Prix de thèse Guy Deniélou 2019 for his PhD thesis. He is a mechanical engineer from the University of Sheffield with a Master's degree in Thermal power and fluids engineering from the University of Manchester in the UK. With more than 8 years of research and project management experience, he has managed and contributed as a researcher in more than 10 EU and other international projects. During his time at Steinbeis EU-VRi, his research interests include risk and resilience management of critical infrastructure with a focus on cyber-physical system security, Digitalization in Industry 4.0, Process/product risk analysis, Multi-criteria decision analysis, and statistical methods in risk analysis.



Marjan Jelić is a software developer with more than 20 years of experience in web, application, and back-end development. Since September 2016, he has worked as a CIO & IT Specialist and Support in the IT department of Steinbeis EU-VRi in Stuttgart, Germany. He is the lead developer of software solutions (tools) in several international projects, mainly in risk and resilience. Marjan Jelić holds a Bachelor of Science degree (B.Sc) in Computer Science from the University of Novi Sad (Faculty of Technical Science in Zrenjanin), Serbia.

Aleksandar Jovanović (MSc. ME, PhD) has worked for industry, research, and EU institutions and universities in Belgium, Italy, Germany, the USA, etc. Since 2001, as the CEO of the Steinbeis EU-VRi (European Institute for Risk & Resilience Management), has provided consultancy in the areas of risk assessment and risk management for industry and the public sector bringing together about 50 industrial and research organizations active in the area of applied risk and resilience management. He is a co-author of the milestone study/book on Future Global Shocks of the OECD (2013), author of 7 books, and over 170 publications. He is the Convener and main author of 3 European CEN standards. He acts as the Liaison Officer in the ISO 31000 (risk management) ISO Committee, as a member of the Committees TC292 (security) and ISO 31010 (risk assessment methods, and as convener for the new ISO 31050 standard.