This is a preprint of the paper submitted and accepted at ECSA Research Paper Track 2024. The final version will be published in the conference proceedings by Springer LNCS.

Modeling and Analyzing Zero Trust Architectures Regarding Performance and Security

Nicolas Boltz*, Larissa Schmid*, Bahareh Taghavi*,
Christopher Gerking, and Robert Heinrich

Karlsruhe Institute for Technology (KIT)
firstname.lastname@kit.edu

Abstract. Zero Trust is considered a powerful strategy for securing systems by emphasizing distrust of all resource access requests. There are different approaches to integrating ZTAs into a system, differing in their components, assembly, and allocation. Early evaluation and selection of the right approach can reduce the costs of resources. In this paper, we propose a novel zero trust architecture (ZTA) metamodel based on literature and industry applications. We introduce our proposed metamodel elements and provide a model instance using the Palladio Component Model (PCM). We describe the requirements for enabling two existing approaches to performance simulation and security data flow analysis on the architectural level and outline how we realize them in our PCM-based implementation. Our evaluation demonstrates the applicability of our ZTA metamodel. It can represent real-world ZTA approaches in various domains, enabling the simulation of performance impact and analysis of the correct implementation of zero trust principles at the architectural level.

1 Introduction

The advancement of the Internet of Things, 5G networks, and cloud technologies has allowed systems and critical infrastructures to shift to more distributed architectures. This shift allows for increased efficiency, for example, by introducing virtual power plants [5] that integrate multiple distributed power-generating components, energy-storing units, and management systems. In the business sector, work from home has become a widely accepted practice, implying that business resources must be accessed off-site and through various devices. Both these trends lead to a larger attack surface of the systems and consequently to an increased number of cyber attacks [20].

A widely adopted approach to coping with such security problems is to separate the system using different access interfaces, firewalls, and intrusion detection systems. However, once an adversary manages to infiltrate the internal domain of the system, they can escalate their privileges and gain access to critical system components and resources. To prevent internal attacks and malicious

* The main authors contributed equally.

access, the zero trust paradigm requires per-request access control for non-public resources, limiting access for unauthorized entities and enforcing granular access control. Contextual factors like the identity of the requesting subject and the environmental context determine access. Factors can be provided by several security mechanisms, like identity and event management systems. Zero trust also upholds the Principle of Least Privilege (PoLP), ensuring subjects are granted only the necessary permissions for resource access, preventing unauthorized propagation within the system. Zero Trust Architectures (ZTAs) incorporate principles of the zero trust cybersecurity paradigm. Existing research shows, that ZTAs can be applied to enhance security in virtual power plants [1], smart healthcare [7], smart manufacturing industries [26], and other more general business domains [7, 16, 38]. While standards regarding ZTA exist [11, 24, 30, 40], an approach that unifies the ZTA structure is missing.

Designing a new system incorporating ZTA or migrating an existing system to use ZTA is not straightforward [36]. The architecture introduces new components and requires additional authentication and authorization checks. How ZTA should be integrated and to what extent is highly dependent on the system under consideration [30]. Moreover, integrating ZTA into the system may lead to overhead due to additional user authentication and authorization, potentially causing performance and availability issues. However, as ZTA covers the entire system, the design and testing of alternatives are complex and may not be viable for systems in active use. Software architecture simulators enable the simulation of a system's quality attributes, such as performance and access control violations, at design-time [6, 28]. However, it is unclear if architecture simulators are applicable to ZTAs and if their predictions regarding quality attributes are accurate.

In this paper, we present our approach to modeling and analyzing ZTAs at the architectural level. We utilize existing technologies to check for performance and security issues. Our main contributions are:

C1 We propose a ZTA metamodel derived from literature and real-world applications.
C2 We provide reusable modeling templates for ZTA architectures.
C3 We enable analysis of multiple quality attributes for our ZTA models.

Our ZTA metamodel (**C1**) aligns different abstract approaches with the terminology provided by the zero trust architecture standard of the National Institute of Standards and Technology (NIST) [30]. The model templates of **C2** are instances of our ZTA metamodel and are created using the Palladio Component Model (PCM) [28]. Compared to other approaches we can simulate the performance impact of integrating the ZTA components into a system, while at the same time analyzing the system for security violations that indicate an improper or insufficient implementation of ZTA.

2 Background

In this section we provide an overview of the background on which our approach is based. It focuses on approaches to modeling and performing analyses, as well as the meaning of ZTA.

2.1 Software Architecture Quality Predictions

Palladio is a tool-supported software architecture simulation approach, that is used to predict a modeled software's Quality of Service (QoS) properties. The Palladio Component Model (PCM) is a metamodel of component-based software architectures [28]. The software architect defines components and their interfaces as reusable system building blocks in the *repository model*. Additionally, the architect describes how instances of the components from the repository model are connected to form a system in the *assembly model*. A software developer uses so-called *Service Effect Specifications* (SEFFs) to provide a coarse-grained execution logic of the services offered by the components in the repository model. The deployment expert adds information about available hardware in the *resource environment model* and describes the deployment of the component instances of the assembly model in the *allocation model*. Different usage profiles and workloads for types of users are defined by a domain expert in the *usage model*. Using this information, Palladio allows for the simulation of a modeled system's behavior under load [4]. Using a variety of measuring points, performance metrics such as response time and hardware utilization can be simulated. Based on these metrics, performance bottlenecks, scalability issues, and reliability threats can be identified and counteracted.

Data Flow Diagrams (DFDs) [12] are unidirectional graphs representing the data flow and processing in systems. Graph nodes are Actors, Processes, or Stores, connected by data flows. DFDs are widely used to analyze various types of security aspects [2, 32, 34, 37] and represent an established way of modeling software architecture. To enable a more general analysis of information security, an extended notation [6, 32] integrates node behavior and labels as first-class entities. Labels represent security-relevant properties, that can be assigned to nodes, and are grouped into label types according to semantic commonalities. Node behavior defines how labels are propagated along the data flow. Security analyses use the node behavior to propagate labels and iterate the resulting DFD to check constraints, e.g., if security-relevant labels can reach certain nodes, or if different labels flow together in a node. In addition, Seifermann *et al.* [32] define model annotations for the PCM, enabling label and behavior definition for various PCM elements, allowing for the extraction of data flows and subsequent security analysis of PCM instances. Building upon these concepts, Boltz *et al.* [6] provide an extensible data flow analysis framework for information security. While these analyses have been successfully applied to check information security properties, like confidentiality and privacy [6, 32], they have not yet been applied to check alignment to zero trust principles.

2.2 Zero Trust Architecture

The National Institute of Standards and Technology (NIST) defines zero trust Architectures (ZTAs) as architectures that apply the concepts of zero trust (see Section 1) in their structure and functionality [30]. To control access to resources, ZTAs contain two core elements: The *Policy Enforcement Point* (PEP) and the

Policy Decision Point (PDP), as shown in Figure 1. The PEP is responsible for intercepting the access request and forwarding the requests to the PDP. It also monitors the access to resources and, if access is not granted, terminates the connection. The PDP evaluates static policies and the context of the request and creates required access credentials.

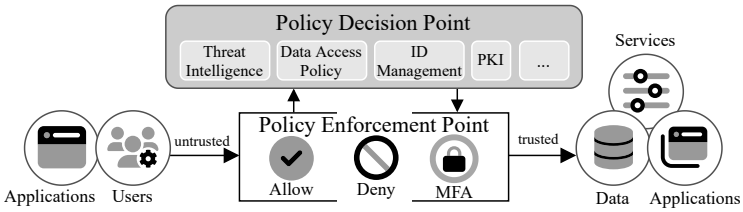


Fig. 1: Overview of ZTA elements [30].

Besides the NIST standard, several other approaches to ZTA exist: Microsoft outlines *zero trust strategies*, which include similar principles regarding zero trust [23]. They also describe logical components of a ZTA, that can be mapped to the PEP, PDP and context providers of the NIST standard. BeyondCorp is a real-world zero trust solution of Google aimed at enterprise applications [38, 16]. Similar to the NIST standard, the BeyondCorp ZTA includes *Data Sources* which provide contextual information, *Access Intelligence components* which handle the access control and policies, *Gateways* such as network switches and web proxies, and the protected resources [25]. The Software Defined Perimeter (SDP) of the Cloud Security Alliance [39, 40] is an approach to granular access control and follows zero trust principles. SDP provides a way to segment the network using software-defined perimeters and hide resources, by the use of policy-based, identity-centric access control to prevent unauthorized access to segments.

3 ZTA Metamodel

We propose a metamodel of ZTA based on literature and real-world approaches [24, 30, 38, 40]. As it provides the most abstract description of ZTA, we align our terminology and overall workflow with the terminology and workflow used in the NIST standard (see Subsection 2.2). Some elements in the NIST standard, such as requests, policies, and contexts, are less formally specified. To be able to capture all relevant information on the architectural level, however, we include them as first-class entities in our model. Figure 2 shows our proposed metamodel.

An access *Request* is sent by a subject that wants to access a *Resource* (1). The *PolicyEnforcementPoint* (PEP) intercepts requests for the resources that the PEP is responsible for (2) and forwards the request to a *PolicyEngine* (PE) of its *PolicyAdministrator* (PA). A PE is responsible for assessing the access request (3). To do so, the PE checks the context of the request based on *Contexts* provided by *ContextProviders* (4). ContextProviders are systems for which different solutions exist, e.g. identity management systems and device databases. These systems are also used outside of a ZTA but are integrated into

a ZTA to provide context information that covers zero trust principles. The NIST standard also defines so-called trust algorithms as special ContextProviders, that provide trust rating to the PE, based on other contexts. *ContextEvaluator* is a specialized ContextProvider that evaluates contexts from other providers before providing a context resulting from the evaluation. An example can be the prior aggregation of multiple contexts. Using the context information, the PE evaluates static *Policies* provided by its available *PolicyProviders* (5) and decides whether the request should be granted. If granted, the PE provides instructions to the PA, including the exact authorization level. The PA is responsible for managing and creating corresponding access credentials for the request, such as access tokens (6). If access is denied, the PA is also responsible for signaling to the PEP to terminate the connection. An extended discussion of the metamodel is provided in the master's thesis [9].

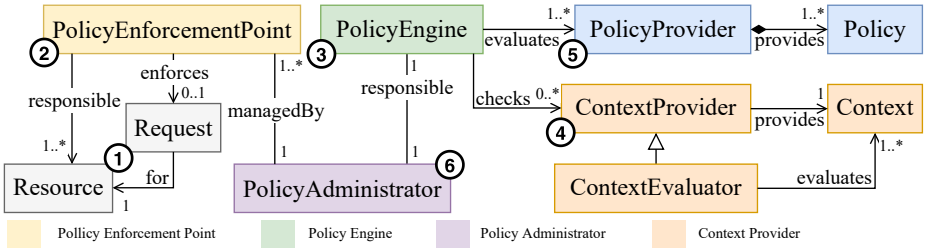
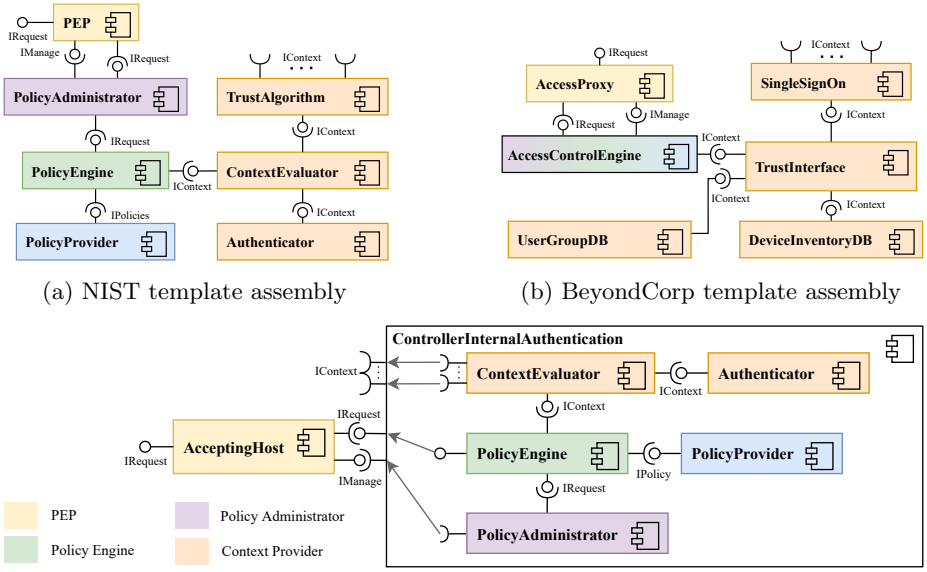


Fig. 2: Structure of our proposed ZTA metamodel.

4 Modeling Templates

To aid software architects in designing or migrating systems with zero trust in mind, we provide instances of our proposed metamodel (see Section 3) that follow three general ZTA approaches. These instances represent reusable modeling templates that include components, interfaces, and some assembly information. We provide templates for three approaches: the NIST standard regarding ZTA [30], the BeyondCorp approach of Google [16, 38], and the Software-Defined Perimeter (SDP) [40] (see Subsection 2.2). We use the PCM (see Subsection 2.1) as a component-based architectural design language for creating the instances. Several elements from our metamodel can be directly mapped to components, while other elements like *Request*, *Context*, and *Policy* are encapsulated in interfaces (e.g. IRequest, IContext, IPolicy) that can be used to connect the components. For the sake of brevity, we only discuss the most important components and interfaces in this description of our templates. Components, interfaces, and data types that are solely required for the technical realization have been omitted. Please refer to our dataset (see Section 9) for a full overview of the PCM instances.

As we have followed the terminology and workflow of the NIST ZTA standard [30] when creating our metamodel, the template directly follows the structure of our metamodel. As shown in Figure 3a, the *PEP*, *PolicyAdministrator*, and *PolicyEngine* are connected by requiring and providing the *IRequest* interface. This interface defines services that, for each component, continue to process an



(a) NIST template assembly

(b) BeyondCorp template assembly

(c) SDP template assembly

Fig. 3: Overview of assembly of modeling template components.

incoming request. The *PolicyProvider* component provides the *IPolicy* interface that the *PolicyEngine* requires. In our NIST standard template, there is always a *ContextEvaluator* component that aggregates context information of authentication and a trust algorithm for the *PolicyEngine* component. The *Authenticator* component represents a subject authentication using some form of credentials. The *TrustAlgorithm* component is also an instance of ContextEvaluator from our metamodel. It aggregates several other contexts from the system. While the kind of context providers and trust calculation logic must be fitted to the system under consideration, we provide some generic components for context providers mentioned in the NIST standard, e.g., device authentication, a security information and event management system, an ID management system, and an activity log system.

The BeyondCorp ZTA defines an access proxy that intercepts requests for resources and an access control engine that manages the requests and access policy checking. The *AccessProxy* is an instance of the PEP of our metamodel and mirrors the *PEP* component in our NIST template. The *AccessControlEngine* component combines the behavior of the *PolicyAdministrator*, *PolicyEngine*, and *PolicyProvider*. In our template, we model this by creating a composite component and encapsulating the corresponding components from our NIST template. In their concrete implementation, the BeyondCorp architecture of Google includes two databases, one for device registration and one for user and role management. In our template, we have added components for both databases as instances of ContextProvider that are connected to the *TrustInterface*. BeyondCorp also uses a single-sign-on system to authenticate subjects using multi-factor authentication. We include a *SingleSignOn* component that is an instance of ContextEvaluator

of our metamodel. By reusing two *Authenticator* components from our NIST template, one for each of its required interfaces, we can model two-factor authentication. The components can be further customized to the specific type of authentication required for the system under consideration. Similar to our NIST templates, contexts are always evaluated by an instance of ContextEvaluator before providing the result to the *PolicyEngine*. In the BeyondCorp ZTA, this component is called *TrustInterface*. It calculates a trust context based on the contexts provided by the connected ContextProviders, similar to the *TrustAlgorithm* in the NIST standard. However, it also takes into account the authentication context of the *SingleSignOn* component.

SDP defines *Accepting Host* (AH), *Initiating Host* (IH), and *Controller* elements. AHs evaluate requests for resources based on policies provided by the *Controller* prior to the request, while IHs represent the subjects making a request. The *Controller* authenticates each requesting IH and deploys access instructions to the AHs of the requested resources. While not following the terminology of the NIST standard, the components and functionality that make up SDP can be mapped to our metamodel and components from the NIST template shown in Figure 3a. Figure 3c shows the allocation of our SDP template. AHs represent the PEPs in the NIST ZTA due to their similar functionality. Since the controller authenticates the IHs and decides which resources to allow access to, the *Controller* represents a combination of PolicyAdministrator, PolicyEngine, and PolicyProvider, similar to the BeyondCorp template. However, the SDP specification includes the authentication as part of the *Controller*. Figure 3c shows the resulting composite component that makes up the *Controller*. To evaluate and authorize the IH, the controller may gather information from other services, such as identity and device management systems, geolocation services, or host validation services. Like for the NIST and BeyondCorp templates, these services represent ContextProviders in our metamodel. As they match, we can simply reuse the context provider components we described for the NIST template. While the SDP template might look more straightforward than the NIST or BeyondCorp templates, one central aspect of SDP is that many AHs are used to define very small perimeters and that each AH is either deployed with the resources it is responsible for or on the direct path to them.

5 Analyzing Quality of Zero-Trust Architectures

Using various analysis tools within a modeling environment involves defining the interaction (tool orchestration) between them. There are different types of orchestration strategies [18], but none of them exactly fits our purpose. Therefore, based on those strategies, we developed a new strategy called Separate Multiple Quality Analysis. This strategy allows tools to run completely independently within a modeling environment to analyze two or more specific quality attributes. In this strategy, the modeling environment provides input to a first analysis tool. By applying a transformation and adding additional requirements to the input, it is then transformed into the inputs of specific tools included in the same workbench. The outputs of these analyses demonstrate the results of different

quality attribute assessments. For performing quality analyses on the software architecture models created with the PCM (see Section 2.1), the system architect needs to assemble the modeled components into an application. Besides this assembly model, the performance simulation and security analysis have additional requirements. The analyses themselves are independent of each other but work on the same model artifacts. In this section, we describe these requirements and how we extend our ZTA metamodel or modeling templates to meet the requirements for the performance analysis (see Section 5.1) and the security analysis (see Section 5.2).

5.1 Enabling Performance Analysis

While our ZTA model templates contain components needed to model ZTA along with their service effect specification (SEFF), component developers need to add resource demands to the specifications to instantiate components for a concrete system and implementation [4]. The SEFFs describe how the provided services of a component relate to its required services, modeling their implementation on an abstract level. The component developers specify the resource usage of internal actions, possibly depending on the parameters the service is called with. How often an internal action is executed can depend on loop iterations and branch transitions. As all of our ZTA components are regular PCM components, the performance simulation fully supports their execution logic.

Additionally to the assembly model, the performance simulation requires a resource environment model, an allocation model, and a usage model. While the resource environment model is not specific to the application, it specifies the hardware used and its processing power, whereas the allocation model specifies where components are deployed. The simulation of the models starts from usage models that contain different usage scenarios, describing how many users are expected to interact with the system in what way, i.e., which services they call.

5.2 Enabling Data Flow Security Analysis

We want to ensure that the security analysis can check that the modeled system aligns with the zero trust principles. As described in Subsection 2.1, the data flow analysis is based on the propagation and annotation of security labels. To align with the principles, we need to define label types and labels that represent if a request is *authorized* and whether a subject of a request is *authenticated* and *trusted*. For authentication, we enable checking of multi-factor authentication by defining the *subject authentication* label type, which contains labels that represent different authentication factors. As it is a common requirement for ZTA [24, 30], we define two labels (*first factor* and *second factor*) to represent 2-factor authentication. When applying our metamodel and modeling templates to other systems with higher authentication requirements, the *subject authentication* label type can be extended or adapted to the needs of the system under consideration. To represent different kinds of authorization, we define label types for the *subject authentication level*, *device authentication*, and *trust*. For the device authentication

and trust label types, we each define the label *authenticated* that represents either the successful authentication of a device or signals a successful trust calculation. We define four abstract authorization levels *Level 0, Level 1, Level 2, Level 3*, each representing a higher authorization level in ascending order. Similarly to the authentication factors, these labels can be adapted to the particular system under consideration, for example, to map to organizational roles for role-based access control [14]. The corresponding labels are used for the label types that represent the rights of the subject (*SubjectAuthorization*), the authorization needed to access a resource (*ResourceRequiredAuthorization*), and the authorization that has been assigned to the request of a subject (*RequestAuthorization*). Using the same labels allows the security analysis to check if a label is set and to compare two labels of different types with each other.

Understanding the behavior of our proposed ZTA components in handling our defined labels is crucial for effective security analysis. The behavior defines whether and how a component alters the labels and, in turn, how they are propagated along the data flow. For the sake of simplicity, we only describe the behavior in natural language. A complete definition can be found in the PCM instance in our dataset (see Section 9). For our ZTA components introduced in Section 3, we define the following behavior: The *DeviceAuthenticator* and *TrustAlgorithm* each alter their respective label type. In our implementation, both set the label *authenticated* if they execute successfully, e.g., when data flows through them. The *Authenticator* components alter the *SubjectAuthentiaction* label type, depending on the used authenticator and the number of authentication factors. The *PolicyEngine* alters the labels of the *SubjectAuthorization* label type. To align with the PoLP, it considers the rights of the subject and dependencies of the requested resource. Most other components do not change labels; they simply forward all labels that flow into them. Our other implementations of ZTA approaches like BeyondCorp [38], and SDP [40] follow the same principles, labels, and label types, based on the mapping described in Subsection 2.2.

6 Evaluation

In this section, we evaluate our approach regarding applicability and ability to simulate performance and security impact of integrating zero trust components into the software architecture. We investigate the following Research Questions:

RQ1 How applicable is our ZTA metamodel when modeling systems based around the principles of zero trust?

RQ2 Can we simulate the performance impact induced by ZTA elements and different execution flows?

RQ3 Can we identify security violations based on principles of zero trust?

We answer RQ1 with a discussion on the applicability of our proposed ZTA metamodel to represent different real-world approaches from related work. For RQ2 and RQ3 we use a Media Store case study from literature. We answer RQ2 by comparing response times of simulations of different scenarios with varying integration of ZTA elements. We answer RQ3 by calculating precision and recall values for different scenarios with manually added violations of zero trust principles.

6.1 Case Study: Media Store

We use the Media Store system [35] that has been widely used for software architecture research [4, 17, 19] to illustrate and evaluate our approach. The Media Store models a file-hosting system to which users can upload and download audio files. It provides basic media management, as well as file encoding and watermarking. The Media Store model does not include any security measures as-is.

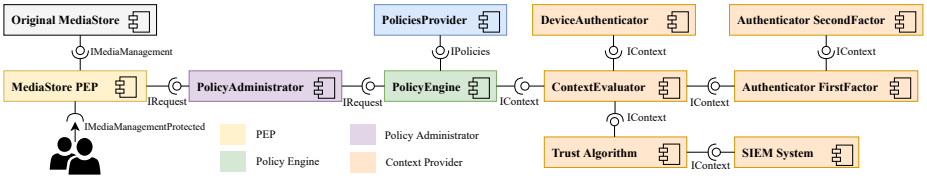


Fig. 4: Overview of the integration of ZTA components into the MediaStore.

We integrate ZTA into the Media Store following the steps presented in the Zero Trust Guide provided by the United Kingdom National Cyber Security Center (NCSC) [24]. Figure 4 shows an overview of our ZTA integration. First, we identify the upload and download functionality and the file access as assets to protect, all provided by the *IMediaManagement* interface of the original *MediaStore*. To protect them, we have to intercept requests to them. We therefore place the *MediaStorePEP* in front of the original *MediaStore* component. To consider the behavior of users when making decisions, we instantiate the *TrustAlgorithm* component that queries data about previous requests by the user from the *Security Information and Event Management (SIEM)* system for decision-making. We instantiate both components from our NIST model template. Next, we integrate policies and policy evaluation into the system. While a *PolicyEngine* is added to evaluate policies, a *PolicyAdministrator* creates the configurations based on the decisions and manages the PEP. We use the *PoliciesProvider* component as policy source for the *PolicyEngine*. Further, we connect a *ContextEvaluator* component to the *PolicyEngine* and supply the device and user authentication as well as the trust algorithm to its required *IContext* roles. As the NCSC guide requires multi-factor authentication for users, we instantiate our *Authenticator* component twice and connect them to each other. Also, we instantiate a *DeviceAuthenticator* component, for device authentication.

We extend the already existing resource environment, consisting of two containers for the *ApplicationServer* and *DatabaseServer*, with a third container for the *ClientDevice*. We allocate all databases to the *DatabaseServer*, the *CredentialsProvider* to the *ClientDevice*, and all other components to the *ApplicationServer*.

6.2 RQ1: Applicability

To evaluate applicability, we focus on two different aspects: First, the ability of our proposed ZTA metamodel to represent different real-world approaches from related work. Second, we discuss the effort needed to integrate ZTA principles in

an existing software architecture model using our ZTA model templates. We also discuss the quality of ZTA after integration using the CISA maturity model [11].
Representation of ZTA approaches using our metamodel: BeyondCorp [38] and SDP [40] presented in Subsection 2.2 represent the most prominent real-world approaches. Especially BeyondCorp seems to have found widespread use, as it is offered as part of the Google Cloud for enterprises [16]. We already show that our metamodel can represent systems based on both approaches with the reusable modeling templates we provide in Section 4. Several other approaches exist in the literature beyond the typical enterprise scenarios for zero trust. Paul *et al.* [26] propose a zero trust model for industrial IoT ecosystems. While they focus on the production domain and, for example, include cyber-physical systems on the architecture level, the approach is largely comparable to BeyondCorp's. Therefore, our metamodel can be easily applied. Ramezanpour *et al.* [27] propose a ZTA for 5G/6G networks that integrates machine learning in some of their proposed components. While this offers new capabilities during runtime, on the architectural level, the introduced 'intelligent' components only represent two new *context providers* when mapped to our ZTA metamodel. Chen *et al.* [8] also propose a software-defined ZTA for 6G networks. Their approach focuses more on the elastic scalability of the architecture while allowing adaptive collaborations among control domains. Similar to the approach of Ramezanpour *et al.* [27], their introduced components for trust calculation, like a Vulnerability Database (VDB), Cybersecurity Event Ledger (CEL) and Anomalous Behaviour Detector (ABD) represent different *context providers* in our ZTA model. Figure 5 shows an excerpt of the resulting architecture using our ZTA metamodel and already defined components of our modeling templates. Lee *et al.* [22] propose situational

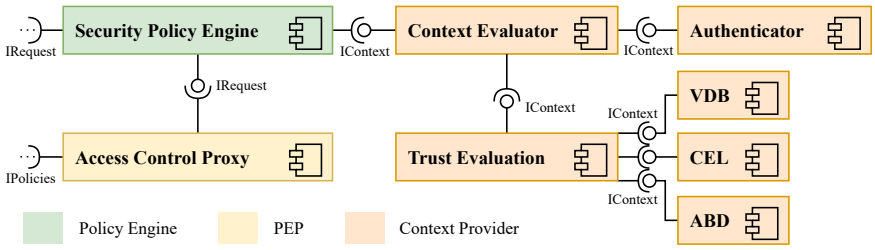


Fig. 5: Excerpt of architecture of Chen *et al.* [8] mapped to our ZTA components.

awareness-based risk adaptable access control for enterprise networks, which follows the general zero trust principles. Their proposed elements can all be mapped to components of our modeling templates, e.g., the *ContextHandler* from the SDP template. Thus our metamodel can be applied. We provide an in-depth description of how we modeled these systems in our dataset (see Section 9).
Integration of ZTA into existing software architecture: We specified a rather broad defense perimeter for our case study (see Subsection 6.1). Compared to the original media store case study, only minimal changes to the system were required to make it compatible with our ZTA components. When trying to define finer defense perimeters, more changes would be required. However, when integrating

a specific overarching concept, like zero trust, into a system that was originally designed without this concept in mind, more extensive changes are to be expected. When assessing the maturity of our implemented ZTA integration based on the CISA Zero Trust Maturity Model [11], a maturity model that defines requirements to achieve increasingly demanding levels of maturity of ZTA-relevant aspects, we conclude that most aspects of our integration have an *initial* or *advanced* rating. A thorough description of our maturity rating and a discussion on how certain aspects can be improved is included as part of the dataset (see Section 9).

6.3 RQ2: Simulating Performance Impact of ZTA Components

We evaluate our model's ability to simulate the ZTA components' performance impact and the different execution flows of the request evaluation process. To get deterministic results, we define the usage of the system so that every user requests the download of four files at a time, each with a size of 40,568,000. We use the Media Store system without ZTA as baseline (S0) and evaluate different scenarios with ZTA. In Scenario S1, all requests are policy-authorized but not context-authorized. This implies that the complete path is executed, with each part of the context being evaluated for every request. Each request should pass the evaluation and be forwarded to the initial Media Store components. In Scenario S2, we turn off usage of the trust algorithm, assuming all subjects accessing the system have the same level of trust.

To obtain accurate performance predictions, component developers need to add resource demands to the model according to the component implementation (see Section 5.1). As we do not add new types of components to the model, we can assume that their simulation behavior is accurate [3]. Therefore, our evaluation focuses on relative differences introduced by the ZTA components rather than the correctness of the absolute response time values. We expect *S1* to show the slowest response times as the configuration of ZTA requires the execution of the complete path for every request, thus adding the most overhead. *S2* should show better performance than *S1* as it does not use the trust algorithm, skipping some parts of the execution process, which results in less overhead and faster response time. Lastly, we expect the baseline *S0* to have the fastest response times as there is no ZTA integrated, and requests are directly passed to the *MediaManagement* component of the MediaStore.

Figure 6 shows the response times of the different evaluation scenarios after simulating their execution for 15.000 time units each. The results confirm our expectations: *S0* is fastest with a median response time of 27.25s. *S1* adds the most overhead, resulting in a median response time of 28.57s. While *S2* still adds overhead to *S0*, it is faster than *S1* with a median response time of 28.38s due to the disabling of the trust algorithm.

6.4 RQ3: Analyzing Security Violations

We evaluate our model's ability to be analyzed for security violations. The data flow analysis of Boltz *et al.* [6] is already able to analyze sufficiently well-specified

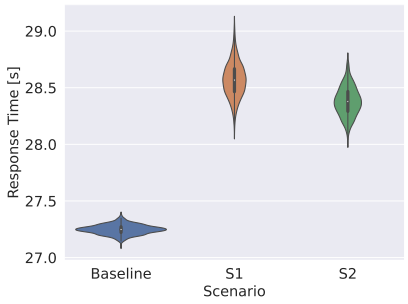


Fig. 6: Performance results of different evaluation scenarios.

models for violations of access control policies. However, zero trust principles go beyond traditional access control, e.g. by enforcing the PoLP. For our evaluation, we focus on violations of zero trust principles that indicate faulty or insufficient implementation of ZTA. Using the labels defined in Subsection 5.2, we define four independent reusable constraints. The constraints cover correct authorization, adherence to the PoLP, subject authentication, and trust calculation.

We use these constraints and compare if the data flow analysis can correctly identify violations. The evaluation is based on a manually created gold standard based on our media store case study system with ZTA. We define a baseline *S0* without violations and four violation scenarios, each containing a violation of a core zero trust principle (see Section 1). As violations, we introduce a flow with unauthorized access (*S1*), a flow violating the PoLP (*S2*), a flow skipping subject authentication (*S3*), and two flows with a lack of trust calculation based on the context (*S4*). Using our constraints, the analysis correctly identified the introduced violations for all scenarios without any false positives. This results in a precision, recall, and F_1 score of 1.0.

6.5 Threats to Validity

We discuss the validity and reliability of our evaluation as characterized by Runeson *et al.* [31]. Our main threat to external validity is the limited generalizability due to the case study-based evaluation. We mitigate this threat by using a well-known case study from literature. Our main threat to internal validity lies in the design decisions about the elements included in our metamodel. We mitigate this threat by basing most of our metamodel on the descriptions of the NIST standard [30]. Regarding construct validity, our evaluation does not comprehensively cover all aspects of zero trust. We cannot fully mitigate this threat but have chosen the examined aspects based on the commonalities and general understanding of the principles described in related work.

7 Related Work

ZTA models have been proposed for various systems, such as for smart industry systems [26], next generation networks [27], and 6G networks based on communities of user equipment [8]. Jung *et al.* [21] propose a ZTA for blocking malicious access to enterprise resources. Three components form the architecture – a PDP,

PEP, and an Authentication Server Function. Lee *et al.* [22] incorporate security situational awareness into Risk Adaptable Access Control. Ghate *et al.* [15] describe an architecture based on automated policy generation definition to achieve low-cost fine-grained network access control. These works primarily discuss how a ZTA could be modeled for various systems and implemented, without considering a model suitable for performance and security analysis.

Furthermore, there is research focusing on the modeling of systems for performance and security analysis. Fernandez *et al.* [13] analyze ZTAs and evaluate security architectures using security patterns. The authors propose a Security Reference Architecture of a ZTA and extract elements from different security patterns to form a concept model of a ZTA. They evaluate some proposed ZTAs by answering questions about its performance and security. The evaluation, however, depends only on experiences since there are no quantitative measures of ZTA systems, and the proposed model is simply a concept. Rodigari *et al.* [29] study the performance of communication between microservices in terms of latency and physical resources with enabling Zero Trust in a multi-cloud environment. Sharma *et al.* [33] employ Discrete Time Markov Chains to model software systems and predict performance, security, and reliability based on its software architecture. Cortellessa *et al.* [10] introduce a framework for modeling the performance and security aspects of software architecture. They create a UML library, which models the basic security mechanisms of encryption, decryption, signature generation, and verification. This is followed by a modeling of composite security mechanisms. Despite some attempts to model ZTA for quality requirements analysis, there is a lack of an approach to modeling a reusable ZTA model while evaluating the impact of integrating the proposed ZTA model into other systems.

8 Conclusion

In this paper, we present our approach to modeling and analyzing ZTAs regarding their performance impact and security. We discuss several principles of zero trust and provide an overview of general approaches to ZTA. Based on these approaches, we devise a ZTA metamodel. By creating instances of our metamodel with the Palladio Component Model (PCM), we provide reusable modeling templates based on the NIST [30], BeyondCorp [38], and SDP [40] architectures. We outline the requirements for enabling PCM performance simulation as well as data flow-based security analysis, and describe how we extend our ZTA model instances to meet these requirements. Based on existing literature, we show that our ZTA metamodel can represent several real-world approaches to ZTA from different domains. Moreover, we show that by using different quality checks, we can simulate the performance impact of integrating ZTA components and identify security problems resulting from an incorrect implementation of ZTA in a system.

9 Data Availability

We provide a data set[1] containing all code artifacts, PCM instances of our ZTA modeling templates, and the used case study model instances.

[1] `https://doi.org/10.5281/zenodo.11580654`

Acknowledgements

This publication is partially based on the research project SofDCar (19S21002), which is funded by the German Federal Ministry for Economic Affairs and Climate Action. This work was also supported by funding from the pilot program Core Informatics at KIT (KiKIT) and the topic Engineering Secure Systems of the Helmholtz Association (HGF), KASTEL Security Research Labs, and the German Research Foundation (DFG) under project number 499241390 (FeCoMASS).

References

1. Alagappan, A., Venkatachary, S.K., Andrews, L.J.B.: Augmenting zero trust network architecture to enhance security in virtual power plants. Energy Reports **8**, 1309–1320 (2022)
2. Alshareef, H. *et al.*: Precise Analysis of Purpose Limitation in Data Flow Diagrams. In: ARES (2022)
3. Becker, M., Becker, S., Meyer, J.: Simulizar: Design-time modeling and performance analysis of self-adaptive systems. (2013)
4. Becker, S., Koziolek, H., Reussner, R.: Model-Based performance prediction with the palladio component model. In: WOSP, pp. 54–65 (2007)
5. Bhuiyan, E.A. *et al.*: Towards next generation virtual power plant: Technology review and frameworks. Renewable and Sustainable Energy Reviews **150** (2021)
6. Boltz, N. *et al.*: An Extensible Framework for Architecture-Based Data Flow Analysis for Information Security. In: ECSA (2024)
7. Chen, B. *et al.*: A security awareness and protection system for 5G smart healthcare based on zero-trust architecture. IEEE IoT Journal **8**(13), 10248–10263 (2020)
8. Chen, X. *et al.*: Zero trust architecture for 6G security. IEEE Network (2023)
9. Cholakov, E.: Modelling and Analysing Zero-Trust-Architectures Regarding Performance and Security, `https://doi.org/10.5445/IR/1000171583` (2024). Master's Thesis.
10. Cortellessa, V. *et al.*: An architectural framework for analyzing tradeoffs between software security and performance. In: Architecting Critical Systems: First International Symposium, ISARCS, pp. 1–18 (2010)
11. Cybersecurity and Infrastructure Security Agency (CISA), CISA Zero Trust Maturity Model, (2023). `https://www.cisa.gov/sites/default/files/2023-04/zero_trust_maturity_model_v2_508.pdf` (visited on 02/23/2024)
12. DeMarco, T.: Structure analysis and system specification. In: Pioneers and Their Contributions to Software Engineering, pp. 255–288 (1979)
13. Fernandez, E.B., Brazhuk, A.: A critical analysis of Zero Trust Architecture (ZTA). Computer Standards & Interfaces **89**, 103832 (2024)
14. Ferraiolo, D.F. *et al.*: Proposed NIST standard for role-based access control. TISSEC **4**(3), 224–274 (2001)
15. Ghate, N. *et al.*: Advanced zero trust architecture for automating fine-grained access control with generalized attribute relation extraction. IEICE Proceedings Series **68**(C1-5) (2021)
16. Google Cloud: BeyondCorp, (2024-03-26). `http://cloud.google.com/beyondcorp`
17. Gorsler, F., Brosig, F., Kounev, S.: Controlling the Palladio Bench using the Descartes Query Language. In: KPDAYS, pp. 109–118 (2013)
18. Heinrich, R. *et al.*: Composing Model-Based Analysis Tools. Springer (2021)

19. Heinrich, R. *et al.*: The palladio-bench for modeling and simulating software architectures. In: ICSE-C, pp. 37–40 (2018)
20. IoT – Market data analysis and forecasts. https://de.statista.com/statistik/studie/id/109209/dokument/internet-der-dinge-market-outlook-report/
21. Jung, B.G. *et al.*: ZTA-based Federated Policy Control Paradigm for Enterprise Wireless Network Infrastructure. In: APCC, pp. 1–5 (2022)
22. Lee, B. *et al.*: Situational awareness based risk-adapatable access control in enterprise networks. arXiv preprint arXiv:1710.09696 (2017)
23. Microsoft Corporation, Evolving Zero Trust, (2021). https://query.prod.cms.rt.microsoft.com/cms/api/am/binary/RWJJdT (visited on 02/23/2024)
24. National Cyber Security Centre UK, ZTA design principles, https://www.ncsc.gov.uk/collection/zero-trust-architecture (visited on 02/23/2024)
25. Osborn, B. *et al.*: Beyondcorp: Design to deployment at google. USENIX Association: ;login: Magazine (2016)
26. Paul, B., Rao, M.: Zero-trust model for smart manufacturing industry. Applied Sciences **13**(1), 221 (2022)
27. Ramezanpour, K., Jagannath, J.: Intelligent ZTA for 5G/6G networks: Principles, challenges, and the role of machine learning in the context of O-RAN. Computer Networks **217**, 109358 (2022)
28. Reussner, R.H. *et al.*: Modeling and simulating software architectures: The Palladio approach. MIT Press (2016)
29. Rodigari, S. *et al.*: Performance analysis of zero-trust multi-cloud. In: 2021 IEEE 14th International Conference on Cloud Computing (CLOUD), pp. 730–732 (2021)
30. Rose, S. *et al.*: Zero Trust Architecture. NIST Special Publication (2020). https://doi.org/10.6028/NIST.SP.800-207
31. Runeson, P. *et al.*: Case Study Research in Software Engineering: Guidelines and Examples. John Wiley & Sons (2012)
32. Seifermann, S. *et al.*: Detecting violations of access control and information flow policies in data flow diagrams. Journal of Systems and Software **184**, 111138 (2022)
33. Sharma, V.S., Trivedi, K.S.: Quantifying software performance, reliability and security: An architecture-based approach. Journal of Systems and Software (2007)
34. Sion, L. *et al.*: Solution-aware data flow diagrams for security threat modeling. In: SAC, pp. 1425–1432 (2018)
35. Strittmatter, M., Kechaou, A.: The Media Store 3 Case Study System. KIT (2016)
36. Teerakanok, S., Uehara, T., Inomata, A.: Migrating to zero trust architecture: Reviews and challenges. Security and Communication Networks (2021)
37. Tuma, K., Scandariato, R., Balliu, M.: Flaws in Flows: Unveiling Design Flaws via Information Flow Analysis. In: ICSA, pp. 191–200 (2019)
38. Ward, R., Beyer, B.: Beyondcorp: A new approach to enterprise security. USENIX Association: ;login: Magazine (2014)
39. WG: SDP and Zero Trust, Integrating SDP and DNS Enhanced Zero Trust Policy Enforcement. CSA (2022). https://cloudsecurityalliance.org/artifacts/integrating-sdp-and-dns-enhanced-zero-trust-policy-enforcement/
40. WG: SDP and Zero Trust, SDP Specification v2.0. CSA (2022). https://cloudsecurityalliance.org/artifacts/software-defined-perimeter-zero-trust-specification-v2/