‘State of the Union’: Evaluating Open Source Zero Trust Components

Tobias Hilbig[1][0000−0002−2904−4758], Thomas Schreck[1][0000−0002−8960−6986], and Tobias Limmer[2][0000−0001−8904−0620]

[1] HM Munich University of Applied Sciences, Munich, Germany
`{tobias.hilbig, thomas.schreck}@hm.edu`
[2] Siemens AG, Munich, Germany
`tobias.limmer@siemens.com`

Abstract. Zero Trust Architecture (ZTA) is a security model based on the principle “never trust, always verify”. In such a system, trust must be established for both the user and the device for access to be granted. While industry adoption of commercial ZTA solutions is accelerating, the state of open-source implementations has yet to be explored. To that end, we survey open-source implementations of zero trust components and put forward a set of ZTA specific requirements to evaluate against. We also identify seven major challenges that hinder the adoption and deployment of open-source zero trust solutions. Our results show that implementations for individual components are much more mature compared to “all-in-one” ZTA solutions. The interoperability between solutions and the development of inter-component protocols are the main areas in which improvements can be made. Despite encouraging developments, we conclude that building ZTAs on top of open-source components is difficult.

Keywords: zero trust architecture · distributed systems security · authentication · authorization

1 Introduction

Network security has always been adapted and improved, both operationally and conceptually, to cope with new requirements and challenges of an ever-changing IT landscape. Despite this, the volume and impact of attacks against IT systems are increasing yearly. Existing perimeter-based network security cannot cope with today’s requirements: Due to work-from-home, cloud computing, and BYOD policies, an organization’s perimeter can no longer be clearly defined and protected. Furthermore, as the software and hardware landscape is becoming more diverse, enforcing strict access control policies in such heterogeneous environments can be challenging.

Zero Trust Architecture (ZTA) is one of the more recent concepts in network security. It can be summarized as “never trust, always verify” [1]. In contrast to the existing perimeter-based network security model, no inherent trust or privileges are granted based on the user’s or device’s physical or logical location.

Instead, a trust algorithm mediates access based on user, device, and service authentication and authorization. ZTA is a promising, significant change in network security philosophy compared with existing approaches and ideas.

While ZTA was conceived almost 20 years ago [2], it took considerable time for the first large-scale implementations. After a significant data breach known as "Operation Aurora" in 2009, Google implemented ZTA and subsequently published an article series [3] called "BeyondCorp" in 2014. Netflix [4] and Microsoft [5] also adopted ZTA in the last few years. Market research [6] shows that large parts of the industry are planning to transition their networks to ZTA in the near to medium-term future. To that end, "ZTA-as-a-service" offers by large enterprises seem to be the primary driving factor. It appears that the availability and maturity of open-source software and protocols necessary for implementing ZTA is an under-researched field.

Therefore, we want to answer the following research questions: (1) What are specific requirements for ZTA components? (2) To what extent do existing and emerging ZTA software solutions and protocols meet these requirements?

Our contributions are: (1) an overview of currently available and emerging ZTA software and protocols, (2) an evaluation of their features and maturity and (3) an analysis of specific requirements for ZTA components. In addition, our results can be used to select suitable solutions for a specific context.

The remainder of this work is structured as follows: We begin by discussing related work in Section 2 and lay out our methodology in Section 3. ZTA components and protocols, together with their requirements, are defined and analyzed in Section 4. Existing and emerging ZTA software and protocols are discussed in Section 5. We evaluate the solutions and present our results in Section 6. Challenges and future work are discussed in Section 7 and the paper concludes with Section 8.

2 Related Work

High-level requirements, or principles, for ZTA are well established and have been discussed extensively in the literature. NIST [1] defines seven tenets for ZTA. Kindervag, in [7], laid out reasons and key concepts for ZTA. ZTA core principles from a business perspective and example use-cases were proposed in a white paper by The Open Group [8]. In a later publication from the same consortium, these principles form the basis for nine high-level commandments for ZTA. While a universally accepted definition of ZTA is yet to be established, consensus for all high-level requirements formulated in these publications emerged.

Requirements have also been discussed for specific use-cases: Rose, in [9], proposes requirements for the implementation of ZTA at federal agencies. Rules for integrating legacy devices into Industrial Control Systems (ICS) based on ZTA were proposed by Køien, see [10]. The literature also provides requirements, challenges and lessons learned regarding real-world ZTA realization. Google published an article series dubbed "BeyondCorp" in 2014, see [3], discussing their approach and application of ZTA in great detail. Similarly, Netflix presented their implementation of ZTA in 2018, see [4]. While specific applications or use-cases can have more nuanced demands, the general principles for ZTA still hold. For this reason, our aim is to formulate and evaluate requirements in an agnostic manner.

Three major surveys of the literature concerning ZTA have been published in the last two years. Yuanhang et al., see [11], conducted a survey of the academic literature, analyzing and comparing ZTA, identity and authentication, access control mechanism, and trust evaluation mechanisms. This work focused on advantages and disadvantages, current challenges and future research trends for ZTA. Buck et al., see [12], did a systematic literature review including gray literature. The authors state that ZTA has been gaining more and more interest in both academia and practice over the last few years, as measured by the number of publications per year. They also state that the literature primarily focuses on conceptual issues and benefits of ZTA, while user-related aspects and possible drawbacks are neglected. Finally, Seyed et al., see [13], surveyed the literature and specifically discussed authentication mechanisms, access control schemes, and encryption in the context of ZTA.

We also reviewed academic work with regard to federation in ZTA, as this concept is highly relevant for ZTA components. The topic was explored by Olson et al., see [14]. In this work, four design objectives for a distributed trust mechanism were given. These design objectives are further broken down into requirements resulting in a federated zero trust architecture in which trust is established via an additional, external proxy component. The concept of Zero Trust Federation (ZTF), together with a proof of concept implementation, was published by Hatakeyama et al., see [15]. Federated operation raises privacy concerns as context information about the user must be exchanged. The ZTF approach allows users to retain control over this information when exchanging with third parties. This is made possible through the use of the Continuous Access Evaluation Protocol (CAEP) [16] and User Managed Access (UMA) [17], an extension for OAuth2.0 [18].

While some publications in the area of zero trust touch the question of requirements for specific components, to the best of our knowledge this is the first academic work that collects and summarizes requirements for each component of a ZTA and evaluates available software solutions and protocols against these requirements.

3 Methodology

All data used in this work was acquired in April 2023. Google Scholar, the BASE database and the backward snowballing technique [19] were employed to collate relevant academic publications. While we did not use a specific search string, we included works in the field of zero trust that discuss general and specific requirements, broad surveys, and federation related publications in the context of zero trust architecture. Gray literature and inaccessible documents were excluded from the results.

Google, Github and the backward snowballing technique were used to find suitable open-source zero trust software solutions and protocols. Our keywords for this search were *zero trust software*, *zero trust implementation* and *open source zero trust*. We excluded abandoned, undocumented and proprietary solutions.

The selection of requirements for components was done as follows: Requirements concerning components were collected from related work, the primary source being the NIST standard. We extended this initial set by adding needed requirements for inter-component communication and federated operation. The resulting set of requirements was then reviewed and discussed individually with each member of our group. This review process was repeated twice, at which point consensus emerged.

4 Architecture and Requirements

For this work, we use NIST's definition of ZTA: "Zero trust architecture (ZTA) is an enterprise's cybersecurity plan that utilizes zero trust concepts and encompasses component relationships, workflow planning, and access policies. [...]" [1]. Moreover, we note that NIST discusses four primary components: The Policy Enforcement Point, the Policy Decision Point, the Policy Information Point, and optionally a client-side agent. These components were initially defined in XACML [20]. They are used to establish trust and mediate access to resources by evaluating authentication, authorization and assurance information for users, devices and services.

Before discussing the components and defining their specific requirements, we reflect upon the general architecture of ZTA and the vital concept of federation between ZTAs. In addition, we discuss the "control plane", i.e., the inter-component communication layer. All requirements listed in the following sections, together with a detailed description, can be found in Appendix A.

4.1 Architecture

ZTA is a fundamentally different approach to network security compared to existing perimeter-based networking. Instead of defining multiple zones with different "trust levels" such as the Internet, a demilitarized zone (DMZ) and the intranet, in ZTA, every asset and the network between assets is considered untrusted by default. In perimeter-based networking, much effort is made to separate these zones, for example by using firewalls and employing physical access control. While ZTA does not attempt to separate assets into zones, a strict separation is done based on the content of transmitted data: Control functionalities, i.e., all communication done between ZTA components, reside on the "control plane", while all other data transfer happens on the "data plane".

The architectural aspects of ZTA have been studied extensively in the past. NIST, in [1], defined four major ZTA types together with requirements for ZTA components, use-cases for ZTA and possible threats. We exclude the "Resource

Portal" and "Device Application Sandboxing" models, as they do not incorporate a client side agent. Instead, we focus on the classic "Enclave" and "Device Agent/Gateway" deployment models. Figure 1 shows a generic ZTA based on the latter model, with ZTA specific components depicted in dark blue. It contains the four main components, i.e., a Policy Information Point (PIP), Policy Enforcement Point (PEP), Policy Decision Point (PDP), an agent, a subject accessing a service via an endpoint and further data sources. In addition to the components themselves, we identified five communication flows between components which are specific to ZTA. They are numbered and discussed in the following sections at the affected components. The remaining, unnumbered flows are not specific to ZTA and can be realized with existing protocols and APIs.

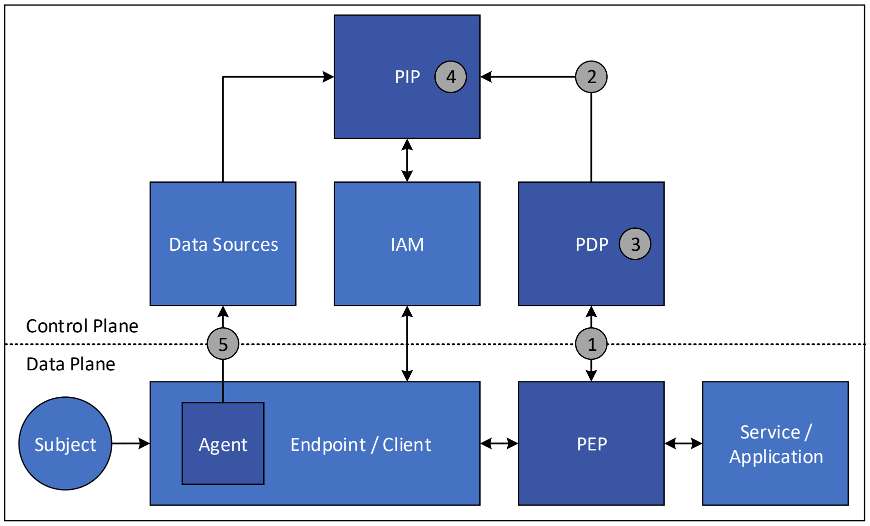

Fig. 1. Generic Zero Trust Architecture with all communication flows

4.2 Federation

In a federated system, multiple distinct entities or organizations collaborate to form a new, larger entity. Trust between participating organizations needs to be established beforehand. In the context of ZTA, federation can be achieved by connecting multiple independent ZTAs to allow seamless data exchange.

User, asset and policy-related information must be exchanged for federated operation. When multiple ZTA's collaborate in this way, that set of information is extended – while all other aspects of ZTA stay essentially the same. PDPs and possibly PIPs need to support this mode of operation, e.g., the sharing of data between parties. From an architectural perspective, ZTA is therefore well suited for federated operation.

4.3 Policy Enforcement Point

The PEP is responsible for enforcing access decisions made by the PDP. From a network perspective, the PEP must be located anywhere between the source and the target of every connection. Moving the PEP closer to the target can be advantageous, as the scope of the trust zone is reduced or even eliminated in case of direct integration in the target application.

We evaluate PEPs based on the following three requirements: (1) The architecture, i.e., integrated into the target application, as a proxy component between client and service, and client-side. (2) The protocol used for interacting with the PDP is also a requirement, see Flow 1 in Figure 1. (3) Push-based demotion and termination of sessions are advanced capabilities a PEP can support and therefore another requirement.

4.4 Policy Decision Point

The PDP is the central component of every ZTA and is responsible for validating and deciding every single access request. Authentication and authorization need to be considered separately here. The PDP needs to authenticate every request, i.e., validating that the requesting party is actually the one it claims to be. This process is usually delegated to Identity and Access Management (IAM) solutions. Authorization is ensured by evaluating the policy. The decision can then be based on authenticated user and, ideally, device information. Usually, services or applications make some authorization decisions themselves. With ZTA, it is possible to move these authorization decisions to the PDP. This approach allows centralized and fine-grained access control schemes to be realized at the PDP.

We evaluate PDPs based on the following five requirements: (1) Supported policy languages, (2) options for ingesting policies, and (3) mechanisms for policy storage are crucial. (4) As a PDP needs to communicate with PEPs, PIPs, and possibly other PDPs, the respective protocols for these purposes are also evaluated, see Flows 1-3 in Figure 1. (5) The last requirement is the support for federated operation.

4.5 Policy Information Point

The PIP is another central component in every ZTA. It collects all data needed for making policy decisions from various data sources. This information is then offered to the PDP in a standardized manner. Therefore, the distinction between PDP and PIP is only functional, allowing the PIP to be possibly integrated into the PDP.

We evaluate PIPs based on the following four requirements: (1) Data sources, i.e., all supported means of acquiring data from external systems and databases.

(2) Since PIPs need to interact with them, supported Identity Providers (IdPs) are another central requirement. (3) The query protocol, i.e., the protocol the PIP offers to PDPs is also relevant, see Flow 2 in Figure 1. (4) PIP-to-PIP communication support, e.g., in distributed or federated environments is the last requirement, see Flow 4 in Figure 1.

4.6 Agent

The agent is installed at the endpoint or integrated into the operating system. It collects relevant information about the host system, such as the operating system version and device trust related data. This information is collected centrally and can be used by the PDP during policy evaluation. It is important to note that some solutions use the agent to establish connections or tunnels to legacy applications or services. This functionality can also be realized via proxies or standalone software and is not necessarily part of the agent nor a requirement. We argue that it is impossible to realize all fundamental ZTA goals in an agent-less ZTA, a view also expressed by NIST, see [1]. Without this component the PIP gains no knowledge about the state of the requesting device, making device trust based decisions infeasible.

We evaluate three requirements for agents: Capabilities for (1) hardware and (2) software collection, i.e., the types of data it can collect about the host system. (3) The protocol used to transmit this data to PIPs or PDPs is the last requirement, see Flow 5 in Figure 1.

4.7 Control Plane

Communication within a ZTA can be logically separated into the “data plane” and the “control plane”. The actual communication is done on the data plane, while management and control functionalities reside on the control plane. ZTA components must therefore support protocols for communication on the control plane. All protocols need to guarantee integrity, confidentiality, and reliability. Non-functional requirements such as performance are out of scope for this work.

To the best of our knowledge, the current protocols for these communication flows are either proprietary or custom-developed and tightly integrated into the respective software solution without ongoing standardization efforts. Due to this tight coupling, the protocols are evaluated as part of the component in question.

5 Implementations and Protocols

Our systematic analysis of ZTA software solutions and protocols suited for usage on the control plane resulted in several implementations and emerging standards that we discuss in the following sections. For each solution, we analyze and discuss use-cases, implemented components, supported features, interfaces, and finally, security properties of employed protocols.

The **Envoy Proxy** project is a proxy component for micro-services, i.e., cloud-native applications [21]. The software is supposed to be installed on the application server and secures communication between applications by tunneling via mTLS connections. Envoy can tunnel any TCP/UDP traffic. In addition, a variety of application protocols are supported, for example, HTTP, Redis, and Postgres. While the primary use-case is securing service-to-service communication, Envoy can also be used as an edge proxy that accepts requests from clients and forwards those to services. The authorization decision for incoming requests can be delegated to external components via custom filters. Envoy can therefore be combined with other zero trust solutions to work as a PEP and can be deployed on Linux, macOS, Windows, and Docker containers. Envoy is licensed under the Apache License 2.0.

OpenZiti offers a complete zero trust solution comprising all necessary components: A PEP (EdgeRouter), a PDP (Controller), a PIP (integrated into the controller), a client software (Client) and a custom developed control plane protocol [22]. Endpoints must use the Client to access applications secured with OpenZiti. EdgeRouters form a mesh and are able to tunnel TCP or UDP-based protocols over untrusted networks. The final EdgeRouter in front of the target application (from a networking point of view) terminates the secure tunnel. In addition to the tunneling mechanism, OpenZiti offers SDKs for C, C#, Swift, and REST. Applications built on top of the SDK can directly interface with the OpenZiti network without needing to terminate the tunnel. The access policy is configured at the Controller. It offers "posture checks" that can be used for authentication and device assurance: (1) operating system type and version, (2) network adapter MAC address, (3) external MFA with a configurable timeout, (4) running applications defined by the path of the executable, and (5) windows domain membership. All connections inside an OpenZiti network are secured using mTLS with X.509 certificates. OpenZiti can be deployed on Linux, Windows, macOS and Docker containers. The client component additionally supports Android and iOS. All components except the client applications for proprietary operating systems are developed as open-source software and licensed under the Apache License 2.0.

Tailscale is a modern zero trust capable VPN solution that can be used to build secure tunnels to services and applications over untrusted networks [23]. Tailscale consists of the client software installed at every host that should be part of the network and the central server software. The clients form a mesh network that securely tunnels traffic. The central coordination server is hosted by Tailscale Inc. and coordinates the distribution of authentication keys and access policies. A third-party open-source implementation of the coordination server called "Headscale" is also available [24]. More complex setups are also possible. For instance, Tailscale is able to construct complete VPN tunnels that encompass all traffic originating from the client, so-called exit nodes. Tailscale is based on Wireguard, a modern VPN protocol and software, see [25]. The client software is available for Linux, BSD, Windows, macOS, iOS, and Android. The coordination server software is available for Linux, BSD, Windows, and macOS.

Tailscale is licensed under the BSD 3-Clause license while the client software user interface code for Windows, macOS, and iOS is proprietary.

Pomerium is an access proxy solution comprised of four components: (1) the proxy service, responsible for tunneling all traffic, (2) the authentication service that connects the IdP to the system and manages session cookies, (3) the authorization service that validates every request against a dynamic access policy, and (4) the "Data Broker Service" that stores session information, identity data, and tokens [26]. The authorization and authentication components check the context of the request, the requesting user's identity and the device identity. Pomerium supports all HTTP-based traffic and allows tunneling arbitrary TCP-based traffic via HTTP. Applications managed by Pomerium receive the user's identity via a signed JWT. The JWT's signature can then be verified by the application. Communication between the components is using gRPC secured with X.509 certificates. Communication between Pomerium and services or applications is secured via mTLS with X.509 certificates. Pomerium is licensed under the Apache License 2.0 and can be deployed on Linux, macOS, and Docker containers.

Boundary is an identity-based access solution consisting of the client software, "Controller" and "Worker" nodes [27]. Both types of nodes can operate redundantly for scaling and failover functionality. Client access requests are authenticated via a controller node, while worker nodes act as proxies for the actual application data. The communication between the service or application and worker nodes is unencrypted, as the secure tunnel is terminated at the worker node. Communication between nodes is based on mTLS secured with X.509 certificates. Boundary can be deployed on Linux, Windows, and macOS and is licensed under the Mozilla Public License 2.0.

Ockam is a library for secure end-to-end communication [28]. It serves as a PEP and can be used to build zero trust capable Rust applications. Key establishment, rotation, and revocation as well as attribute-based access control mechanisms are supported. Ockam supports TCP, UDP, Websockets and Bluetooth as transport protocols. Ockam applications establish secure channels via the Noise Protocol Framework or the X3DH protocol [29]. Ockam is licensed under the Apache License 2.0.

Oathkeeper is a zero trust capable HTTP reverse proxy with an integrated decision API [30]. It can be used as a combined PEP and PDP while also supporting PDP functionality in standalone mode. The reverse proxy ensures that requests satisfying the access rule pipeline are forwarded to the upstream server. The access rule pipeline consists of four components: (1) Authentication handlers inspect and validate HTTP traffic using sessions (cookie-based), tokens, OAuth 2.0 or JWTs. (2) Authorization handlers check access permissions based on Ory Keto or arbitrary remote HTTP and JSON endpoints. (3) Mutation handlers can be used to transform and augment authentication information into formats the authentication backend understands. This includes creating signed JWTs, arbitrary HTTP header data and cookies. Authentication information can also be enriched by querying external APIs. (4) Error handlers that define behavior in case authentication or authorization fails. Options include JSON responses, HTTP redirects, and HTTP 401 “WWW-Authenticate” responses. The access control API can also be connected to Ambassador, the Envoy proxy, AWS or Nginx.

Keto, see [31], is an open-source implementation of Zanzibar [32], an authorization system developed by Google. It can be used as a PDP. Keto stores the policy as relation tuples between subjects and objects. Relation tuples form a graph from which permissions can be deducted. Relations can be queried via HTTP and gRPC API endpoints, allowing read operations to check permissions, query relations, and list objects. Write operations can be used to modify, insert and delete objects and relations. Access to Oathkeeper and Keto APIs is secured using HTTPS. Both solutions can be deployed on Linux, macOS, Windows, and FreeBSD and are licensed under the Apache License 2.0.

The **Open Policy Agent** (OPA) is a general-purpose, open-source policy engine [33] that can be used as a PDP. Together with the declarative “REGO” policy language, it allows the creation and execution of fine-grained policies that can be used to build ZTAs. OPA offers an HTTP REST API for evaluating policies that returns JSON data. In addition, OPA can be integrated into Go applications via an SDK. Policies can also be compiled into WebAssembly instructions and can be embedded into any WebAssembly runtime. OPA can be deployed on Linux, macOS, Windows, and via Docker containers. OPA is licensed under the Apache License 2.0.

The **Secure Production Identity Framework For Everyone** (SPIFFE) and the accompanying **SPIFFE Runtime Environment** (SPIRE) are a standard and reference implementation for identification, attestation and certificate distribution in dynamic software systems [34]. The SPIFFE standard defines SPIFFE IDs that can be used as identities for services. These IDs can be encoded into SPIFFE Verifiable Identity Documents (SVIDs), cryptographically verifiable documents, i.e., certificates. SPIFFE also defines an API for issuing and retrieving SVIDs called the “Workload API”. SPIRE implements SPIFFE and offers additional features. SPIRE consists of agents installed alongside the application or service and a server component. SPIRE can perform node and workload attestation and registration as well as rotate keys and certificates. It also provides services with access to secret stores and databases and enables federation of SPIFFE systems across trust boundaries.

SPIREs API endpoint is offered by the agent and used by a workload for creating and validating SVIDs. Communication uses gRPC over a Unix Domain or TCP socket. As SPIFFE is often used to establish a root of trust, TLS must not be required by implementations. The communication between the agent and the central server component is secured using mTLS with a pre-shared “bootstrap bundle”.

In the context of ZTA, SPIFFE / SPIRE can be used as part of the control plane. It can be used to quickly and securely retrieve, rotate, and manage service identities and corresponding keys in cloud environments. In this context, containers for services are usually rapidly deployed and decommissioned. This is the primary use-case for SPIFFE / SPIRE, as requirements differ from traditional mechanisms for identity management of client devices and servers. It is licensed under the Apache License 2.0.

The **Shared Signals Framework** (SSF) is currently being developed by the "Shared Signals Working Group" at the OpenID foundation and aims to standardize a security event sharing protocol [35]. The OpenID foundation plans to develop and provide a reference implementation for SSF to facilitate interoperability testing. Two profiles have been developed for SSF: The Continuous Access Evaluation Protocol (CAEP) and the Risk Incident Sharing and Coordination (RISC). **CAEP** was proposed by Google [16] and later merged into SSF. Without Continuous Access Evaluation, access decisions are only made once before establishing connections. To fully realize the benefits of ZTA, access decisions must be evaluated continuously. To that end, CAEP standardizes events for communicating access property changes between zero trust components. For example, the "Device Compliance Change"-Event can signal that a device no longer fulfills an organization's security policy, possibly including a reason. Such an event could be generated by an agent, received by a SIEM, and later taken into account during policy evaluation at the PDP. **RISC** is the second profile in development. It focuses on transmitting events and information concerning user account security. While not an integral part of ZTA, it can be used for collaboration in federated scenarios. Events have been specified to signal changes related to accounts, credentials and recovery information. For example, RISC can be used to prevent attackers from using compromised credentials several times at different providers. As SSF is built upon Security Event Tokens (SET), see [36], TLS 1.2 or higher must be supported for the transport of events. In addition, SETs must be encrypted in case they contain personally identifiable information (PII) and must ensure integrity, for example by using JWS [37].

Other solutions and commercial offers were found during our search but not included in the evaluation: "beyond" [38] and "helios" [39] are zero trust HTTP access proxy solutions. Due to missing documentation and stalled development, these solutions were not considered. "TRASA", see [40], is an identity and context aware access proxy. It can be used to secure remote access to internal services. TRASA supports RDP, SSH, HTTP, and MySQL. Since the development of TRASA has stopped in December 2021, it was excluded from the evaluation. "Pritunl Zero", see [41], is a zero trust proxy solution for SSH and HTTP connections. While under active development, this software is licensed under a custom license that only allows non-commercial use and forbids the distribution of derivative works. It primarily serves the use-case of centralizing the management of SSH keys through a custom-built certificate authority and a custom SSH client. The offered authentication options are limited, and the solution supports neither device authentication nor assurance. The developers offer a commercial and proprietary solution called "Pritunl" with extended features and support. For these reasons, Pritunl Zero was excluded from the evaluation.

In addition to the open-source solutions described and discussed in the previous sections, we discovered multiple commercial zero trust offers. Google (BeyondCorp Enterprise) and Microsoft (integrated into Microsoft 365) are the primary vendors in this context. Both are cloud-based and offer advanced solutions for zero trust security. These solutions are proprietary, hosted and operated by third parties. Therefore, their capabilities cannot be evaluated like it is the case with open-source software and are of lesser academic interest. Finally, NIST is working on a practice guide titled "Implementing a Zero Trust Architecture" in cooperation with industry partners, see [42]. The goal is to introduce a reference ZTA built on commercially available technology.

6 Evaluation

A comparison of the nine ZTA software systems analyzed in this work is shown in Figure 2. It is based on the generic ZTA constructed in Section 4. The implemented components are highlighted for every software system investigated in this work. The following evaluation is done on a per-component basis for the requirements defined in Section 4 and listed in Appendix A. General findings and directions for future work are discussed in Section 7.

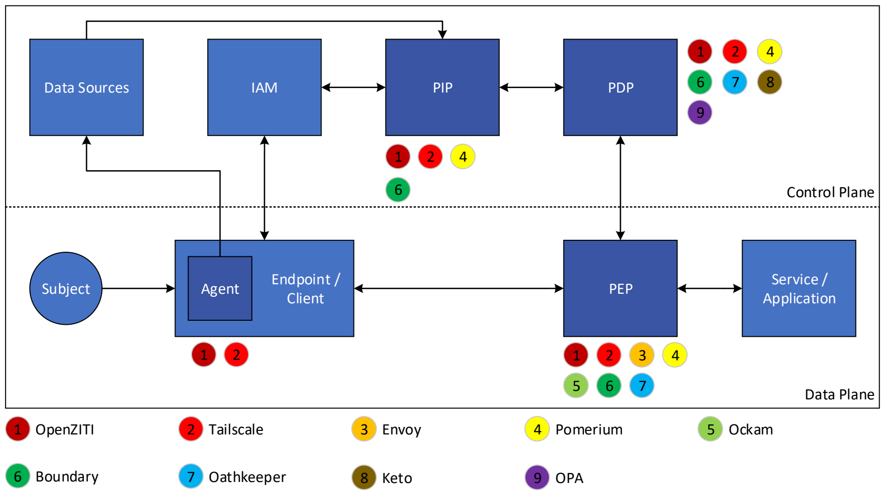

Fig. 2. Component-wise comparison of ZTA software

6.1 Policy Enforcement Point

Existing solutions support all three architectures for PEPs. Support for proxy mode is implemented in all solutions except Ockam, a library that can be used to build zero trust capable applications. Integrated and client-side PEP operation is only provided by two solutions. This result can be attributed to the fact that many solutions build tunnels for the transmission of data plane and inter-component communication. The protocol to connect PEPs to PDPs is self-developed in nearly all solutions. This prevents interoperability, i.e., mixing components from different vendors. The only exception is the Envoy proxy project, a standalone PEP implementation with vast filtering and processing capabilities for requests. Envoy can query PDPs via gRPC and standalone HTTP calls, making it the most capable solution in this regard. Finally, push-based demotion or termination of connections is not documented for any solution. We assume that this advanced capability was not prioritized in the past. We conclude that existing PEPs partially meet the requirements we defined, although interoperability and non-tunnel based operation is lacking.

6.2 Policy Decision Point

The primary requirement for PDPs is the language in which policies are expressed. Only the Open Policy Agent uses a well-defined language for this purpose (REGO), while all other solutions implement custom languages. It is unclear why most examined solutions do not implement established policy languages such as XACML. Policies can be ingested via various means in all solutions, with CLI-based configuration being the most common method. We note that none of the solutions allow querying external databases, which would be ideal for large-scale and distributed ZTA environments. Similarly, only two solutions, Pomerium and Boundary, support external storage of configured policies, with the remainder supporting local storage only. The protocol for PDP-to-PIP communication is custom developed by OPA, Boundary and Pomerium. In all other solutions, the PIP is either part of the PDP itself or querying PIPs is unsupported. For PDP-to-PEP communication, Oathkeeper and OPA implement a custom-developed REST API, while Keto supports HTTP and gRPC APIs. All other solutions use custom, non-interoperable protocols. Finally, PDP-to-PDP communication, and therefore federated operation, is not supported by any of the evaluated software systems. To conclude, existing PDPs are more mature than PEPs, especially the standalone OPA, which is build for this exact purpose. However, self-developed policy languages and inter-component communication protocols are areas in which the requirements are not properly met.

6.3 Policy Information Point

We were unable to locate standalone implementations for PIPs. Instead, the examined solutions implement the PIP as part of the PDP. This can serve as an explanation to why no solutions supports external data sources apart from Tailscale, Headscale, and Pomerium, which are able to use authentication data from IdPs. They support a vast number of IdPs, among them G Suite, Azure, and generic OIDC or SAML providers. In addition, all solutions use an internal RBAC model for configuring an access policy. None of the solutions support a protocol for connecting to PDPs, because they merge the PDP and PIP components allowing data exchange to happen internally. For the same reason, federated operation is also unsupported in all solutions. This allows us to conclude that the requirements for PIPs are not fulfilled in any solution.

6.4 Agent

Data collection about the host system's software and hardware is the main functionality for agents. None of the solutions fulfill these requirements in a strict sense, i.e., collecting and transmitting this data to an external component such as a PIP. Instead, agent implementations in Boundary, OpenZiti, and Tailscale establish tunnel connections to virtual overlay networks which terminate at PEPs. OpenZiti is the only solution with rudimentary device assurance checks during connection establishment, e.g., validating operating system type and version. This also results in custom developed protocols for Agent-to-PIP communication in solutions that implement this feature. Microsoft's zero trust solution encompasses an agent component called "Intune" that is integrated in the Windows operating system. It is a notable exception with regard to the data collection requirement, as vast information about the state of both software and hardware of the host can be collected and evaluated centrally. As Intune is a proprietary, commercial solution, it can not be used in combination with third party ZTA components. We conclude that in a strict sense, no open-source implementations of agents exist, and therefore, none of the requirements are fulfilled.

7 Challenges and Future Work

In Section 5 we discussed the various aspects of current software solutions and protocols for ZTA. The requirements for the evaluation were explained in Section 4 and matched to the solutions in Section 6. Based on our detailed analysis, we can identify the different challenges we have observed and formulate directions for future work in this field. In this section, we discuss them in detail.

Interoperability: Our current networks are based on open standards that allow communication between individual solutions. This interoperability was crucial for the success of many network technologies. In the field of ZTA, we currently see many proprietary solutions that do not allow this interoperability between implementations. Complete solutions that implement all four primary ZTA components such as OpenZiti, Tailscale, and Pomerium offer few or no means to interoperate with other solutions. Partial solutions such as Envoy or OPA implement specific components and are better positioned in this regard by offering custom API endpoints. Nevertheless, interoperability is a significant challenge with existing solutions and a promising direction for future work. Standards for inter-component communication need to be standardized and implemented.

Control Plane: We assess that no open standards or protocols exist specifically for communication between ZTA components on the control plane. This applies to both communication between individual components deployed as part of the same ZTA, for example, between a PEP and the PDP, and inter-organizational communication in the federated scenario, i.e., between PDPs. Promising standardization efforts such as SSF are ongoing. SSF aims to address inter-organizational and inter-service security event sharing. This is an essential step towards improving the overall security posture of IT systems and not limited to ZTA. These protocols and standards need to be developed in the future. Research in this direction can foster the adoption of ZTA by allowing cross-vendor component compatibility.

Federation: As discussed in Section 4.2, ZTA is an ideal candidate for federated operation. We notice that none of the solutions we assessed support federation, which is not surprising given the lack of suitable control plane protocols. Academic work concerning this topic is also sparse. As the adoption of ZTA progresses, the importance of federated operation will rise accordingly. We therefore see the possibility for future work in federated architecture, policy evaluation in federated systems, and privacy-related aspects.

Device Trust and Assurance: Existing solutions are well-positioned with regard to user authentication. However, device identification, trust, and assurance are severely lacking, as none of the solutions support an agent with the necessary capabilities. Including information about client devices in the decision process of the PDP is one of the main tenets of ZTA. This provides a variety of directions for future work. For example, research can focus on how to acquire this data in specific scenarios such as enterprise IT or cloud setups. Extending this idea to diverse environments with limited hardware capabilities, such as the IoT or OT space, is another possible direction. Protocol-based support can also be considered, for example by integrating device identification and assurance checks in standard authorization protocols such as OpenID Connect [43].

Policy Information Point: While PIPs as a concept are well-defined and referenced by the literature, not a single standalone open-source PIP exists today. Complete ZTA solutions offer PIPs with rudimentary functionality, falling short of what is required by a wide margin. Specifically, interfacing with external data sources other than IdPs, e.g., SIEMs or CTI systems, is not supported by any solution. This hinders the adoption of ZTA as a PIP offers functionality central to the idea of ZTA. Future work should examine why this is the case and focus on developing solutions.

Agent: Challenges related to the agent component are similar to PIPs. First, standalone agent components do not exist. Instead, they are implemented as part of a larger solution as it is the case with Tailscale or OpenZiti, offering only basic agent functionality. Microsoft offers an agent that is integrated into the Windows operating system as part of their ZTA solution. We believe this is the right approach to the problem since agents running as standalone components in user space suffer from several drawbacks. For adequate visibility and control functionality, they must run with elevated privileges and interface with the operating system to acquire the necessary data. It would therefore be ideal to integrate this functionality directly into the operating system. This task relies on the vendor for proprietary systems such as Windows, macOS and iOS.

Future work can focus on how to securely acquire and collect this information in open systems such as Linux and BSD derivatives, for example, by integrating it directly into the kernel.

Scaling: One crucial factor in deploying ZTA within an organization's IT network is the topic of scaling. IT infrastructures must be globally reachable with low latency and capable of handling a large number of user sessions simultaneously. Of the examined solutions, six support standalone operation only. Therefore, we propose to look into this functionality in current and future implementations.

8 Conclusions

Zero Trust Architecture is a network security paradigm in which trust, and thus access, is explicitly granted based on user and device authentication. As an organization's perimeter can no longer be clearly defined and protected, the location of the user and the device does not impact authorization decisions within ZTA. While is an emerging field of research with accelerating, commercial deployment and usage, the state of open-source solutions is not well understood. In this work, we surveyed available open-source ZTA solutions and discussed requirements for ZTA software components. Based on these, we evaluated how mature the open-source software components are.

Our findings show that some solutions in the zero trust space aim to implement an “all-in-one” solution, i.e., all components necessary to deploy a ZTA. In contrast to these, there are standalone components that are capable of interfacing with third-party software, allowing modular ZTAs to be built. With regard to protocols, the “Shared Signals Framework” standardization effort stands out as a major development for an open control plane protocol. We identified seven key challenges that offer the potential for future work: Component interoperability, control plane protocols, and federated ZTA are still in the early stages. Concerning components, we assess that standalone Policy Information Points and agents are not available as of today. Device trust and assurance functionalities are essential to ZTA but need to be improved going forward, as existing solutions barely support them. Lastly, open questions regarding scaling in ZTA also need to be addressed.

To conclude, we assess that in general, open-source solutions for ZTA are not yet mature. Protocols for ZTA are even less developed than the software side.

Data availability

The full results of the evaluation, i.e., a table listing the components together with our assessment of fulfilled requirements, is available online [44].

Competing interests

All authors declare that they have no conflicts of interest.

References

1. National Institute of Standards and Technology: [NIST SP 800-207] Zero Trust Architecture. NIST Special Publication - 800 series (2020)
2. T.J. Forum and H. World: Jericho Forum ™ Commandments. Forum American Bar Association (2007)
3. R. Ward and B. Beyer: Beyondcorp : a new approach to enterprise security. ;login:: the magazine of USENIX & SAGE 39(6), 6–11 (2014)
4. B. Zimmer: LISA: A Practical Zero Trust Architecture. In: Enigma 2018 (Enigma 2018). USENIX Association, Santa Clara, CA (2018)
5. Microsoft Corporation: Zero Trust Model - Modern Security Architecture | Microsoft Security, (2022). `https://www.microsoft.com/en-us/security/business/zero-trust` (visited on 04/13/2023)
6. Microsoft and Hypothesis Group: Zero Trust Adoption Report. (2021)
7. J. Kindervag: No More Chewy Centers : Introducing The Zero Trust Model Of Information Security. (2010)
8. The Open Group: Zero Trust Core Principles, (2021). `https://publications.opengroup.org/w210`
9. S. Rose: Planning for a Zero Trust Architecture : A Planning Guide for Federal Administrators. (2022)
10. G.M. Køien: Zero-Trust Principles for Legacy Components: 12 Rules for Legacy Devices: An Antidote to Chaos. Wireless Personal Communications 121, 1169–1186 (2021). DOI: `10.1007/s11277-021-09055-1`
11. Y. He, D. Huang, L. Chen, Y. Ni, and X. Ma: A Survey on Zero Trust Architecture: Challenges and Future Trends. Wireless Communications and Mobile Computing 2022 (2022)
12. C. Buck, C. Olenberger, A. Schweizer, F. Völter, and T. Eymann: Never trust, always verify: A multivocal literature review on current knowledge and research gaps of zero-trust. Computers and Security 110, 102436 (2021). DOI: `10.1016/j.cose.2021.102436`
13. N.F. Syed, S.W. Shah, A. Shaghaghi, A. Anwar, Z. Baig, and R. Doss: Zero Trust Architecture (ZTA): A Comprehensive Survey. IEEE Access PP, 1 (2022). DOI: `10.1109/ACCESS.2022.3174679`
14. K. Olson and E. Keller: Federating trust: Network orchestration for cross-boundary zero trust. Proceedings of the 2021 SIGCOMM 2021 Poster and Demo Sessions, Part of SIGCOMM 2021 (2021). DOI: `10.1145/3472716.3472865`
15. K. Hatakeyama, D. Kotani, and Y. Okabe: Zero Trust Federation: Sharing Context under User Control towards Zero Trust in Identity Federation. 2021 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events, PerCom Workshops 2021 (2021). DOI: `10.1109/PerComWorkshops51409.2021.9431116`
16. A. Tulshibagwale: Re-thinking federated identity with the Continuous Access Evaluation Protocol | Google Cloud Blog, (2019). `https://cloud.google.com/blog/products/identity-security/re-thinking-federated-identity-with-the-continuous-access-evaluation-protocol` (visited on 04/13/2023)
17. E. Maler, M. Machulak, J. Richer, and T. Hardjono: User-Managed Access (UMA) 2.0 Grant for OAuth 2.0 Authorization. Tech. rep., (2019)
18. D. Hardt: The OAuth 2.0 Authorization Framework. RFC 6749, RFC Editor (2012)

19. C. Wohlin: Guidelines for snowballing in systematic literature studies and a replication in software engineering. In: Proceedings of the 18th international conference on evaluation and assessment in software engineering, pp. 1–10 (2014)
20. A. Anderson, A. Nadalin, B. Parducci, D. Engovatov, H. Lockhart, M. Kudo, P. Humenn, S. Godik, S. Anderson, S. Crocker, *et al.*: eXtensible Access Control Markup Language (XACML) Version 2.0. Oasis (2004)
21. Envoy Project Authors: envoyproxy/envoy: Cloud-native high-performance edge / middle / service proxy, (2016). `https://github.com/envoyproxy/envoy` (visited on 04/13/2023)
22. Styra, Inc.: OpenZiti: programmable network overlay and associated edge components for application-embedded, zero-trust networking, (2019). `https://github.com/openziti/` (visited on 04/13/2023)
23. J. Fond: Juanfont/Headscale: An Open Source, Self-Hosted Implementation of the TAILSCALE Control Server, (2020). `https://github.com/juanfont/headscale` (visited on 04/13/2023)
24. Tailscale, Inc.: Tailscale is a WireGuard-based app that makes secure, private networks easy for teams of any scale. (2020). `https://github.com/tailscale` (visited on 04/13/2023)
25. J.A. Donenfeld: Wireguard: next generation kernel network tunnel. In: NDSS 2017, pp. 1–12. The Internet Society (2017)
26. Pomerium, Inc.: pomerium/pomerium: Pomerium is an identity-aware access proxy. (2019). `https://github.com/pomerium/pomerium` (visited on 04/13/2023)
27. HashiCorp, Inc.: hashicorp/boundary: Boundary enables identity-based access management for dynamic infrastructure. (2020). `https://github.com/hashicorp/boundary` (visited on 04/13/2023)
28. Ockam, Inc.: build-trust/ockam: Orchestrate end-to-end encryption, mutual authentication, key management, credential management & authorization policy enforcement — at scale. (2018). `https://github.com/build-trust/ockam` (visited on 04/13/2023)
29. M. Marlinspike and T. Perrin: The X3DH Key Agreement Protocol. Open Whisper Systems 283, 10 (2016)
30. Ory Corp: ory/oathkeeper: A cloud native Identity & Access Proxy / API (IAP) and Access Control Decision API that authenticates, authorizes, and mutates incoming HTTP(s) requests. (2017). `https://github.com/ory/oathkeeper` (visited on 04/13/2023)
31. Ory Corp: ory/keto: Open Source (Go) implementation of "Zanzibar: Google's Consistent, Global Authorization System". (2018). `https://github.com/ory/keto` (visited on 04/13/2023)
32. R. Pang, R. Caceres, M. Burrows, Z. Chen, P. Dave, N. Germer, A. Golynski, K. Graney, N. Kang, L. Kissner, J.L. Korn, A. Parmar, C.D. Richards, and M. Wang: Zanzibar: Google's Consistent, Global Authorization System. In: 2019 USENIX Annual Technical Conference (USENIX ATC '19), Renton, WA (2019)
33. Styra, Inc.: open-policy-agent/opa: An open source, general-purpose policy engine. (2015). `https://github.com/open-policy-agent/opa` (visited on 04/13/2023)
34. The SPIFFE authors: SPIFFE: Secure Production Identity Framework for Everyone, (2017). `https://spiffe.io` (visited on 04/13/2023)
35. OpenID Foundation: Shared Signals – A Secure Webhooks Framework | OpenID, (2017). `https://openid.net/wg/sharedsignals/` (visited on 04/13/2023)
36. P. Hunt, M. Jones, W. Denniss, and M. Ansari: Security Event Token (SET). RFC 8417, RFC Editor (2018)

37. M. Jones, J. Bradley, and N. Sakimura: JSON Web Signature (JWS). RFC 7515, RFC Editor (2015)
38. cogolabs contributers: cogolabs/beyond: BeyondCorp-inspired HTTPS/SSO Access Proxy. Secure internal services outside your VPN/perimeter network during a zero-trust transition. (2017). `https://github.com/cogolabs/beyond` (visited on 04/13/2023)
39. C. Yakimov: cyakimov/helios: Identity-Aware Proxy, (2019). `https://github.com/cyakimov/helios` (visited on 04/13/2023)
40. Seknox Pte. Ltd.: seknox/trasa: Zero Trust Service Access, (2020). `https://github.com/seknox/trasa` (visited on 04/13/2023)
41. Pritunl, Inc.: pritunl/pritunl-zero: Zero trust system, (2017). `https://github.com/pritunl/pritunl-zero` (visited on 04/13/2023)
42. A. Kerman, M. Souppaya, P. Grayeli, and S. Symington: Implementing a Zero Trust Architecture (Preliminary Draft). Tech. rep., National Institute of Standards and Technology (2022)
43. N. Sakimura, J. Bradley, M. Jones, B. De Medeiros, and C. Mortimore: OpenID Connect Core 1.0. The OpenID Foundation (2014)
44. T. Hilbig: hm-seclab/paper-th-zta-components-materials: Supporting materials for STM 2023, (2020). `https://github.com/hm-seclab/paper-th-zta-components-materials` (visited on 08/19/2023)

A List of requirements

In the following, we list the requirements used in the evaluation together with a short explanation, examples and the desired state.

A.1 Policy Enforcement Point

- **Architecture:** *Integrated operation*, i.e., directly integrated into the target application or service, for example via libraries. *Proxy mode*, i.e., as a seperate component in front of the server. *Client side*, i.e., the PEP is deployed on the client device. All architectures are equally desirable.
- **Supported protocols for PEP-PDP communication:** A list of protocols the PEP supports for interacting with PDPs. Existing, well-defined protocols with wide usage are desirable.
- **Push-based demotion and termination:** This advanced feature allows PDPs to demote or terminate established sessions by instructing the PEP to do so. Support for this is desirable.

A.2 Policy Decision Point

- **Supported policy languages:** A list of languages the PDP supports for encoding policies. Existing, well-defined languages with wide usage are desirable.
- **Options for ingesting policies:** A list of options allowing the ingestion of policy information. Examples include user interfaces, REST APIs and the file system. CLI- or API-based ingestion is preferred.

– **Policy storage mechanisms:** A list of supported ways to store policies. The local filesystem is an example. Here it is desirable to have the option of using a database.
– **Supported protocols for PEP, PIP, PDP communication:** A list of protocols the PDP supports for interacting with other components. Existing, well-defined protocols with wide usage are desirable.
– **Federated operation:** The capability and maturity of operating the PDP in a federated environment. It is desirable to have this feature.

A.3 Policy Information Point

– **Data sources:** A list of supported data sources the PIP can query. Examples include device registries, generic databases and CTI feeds. Support for many sources is desirable.
– **Identity providers:** A list of IdPs the PIP can interface with, for example generic support for SAML. Support for well known IdPs and protocols is desirable, especially OIDC.
– **Query protocol:** The protocol, or the protocols, the PIP supports for querying data. This protocol can then be used by the PDP for policy decisions. Existing, well-defined protocols with wide usage are desirable.
– **Supported protocols for PIP-PIP communication:** In federated environments, PIPs might need to interface with other PIPs to exchange data. Existing, well-defined protocols with wide usage are desirable.

A.4 Agent

– **Hardware-based collection capabilities:** A list of information items the agent is able to collect about the hardware of the client. Examples include the secure boot state, firmware version information or CPU vulnerabilities. It may be desirable to collect as much information as possible.
– **Software-based collection capabilities:** A list of information items the agent is able to collect about the software of the client. Examples include the operating system type and version, currently running software or antivirus software state. It may be desirable to collect as much information as possible.
– **Supported protocols for Agent-PIP communication:** A list of protocols the agent supports for interacting with PIPs. Existing, well-defined protocols with wide usage are desirable.