Evaluating Zero Trust Mechanisms in a Virtualized Laboratory Environment

Baktybergenova Aruzhan, Sherzodbek Saydakhmedov,Ibrayev Daulet
SITAM /SDU University· Kaskelen, Kazakhstan
{230103218,230103382,2301033028}@sdu.edu.kz

Abstract—Zero Trust Architecture (ZTA) is a cybersecurity approach based on the principle of "never trust, always verify." This study reviewed major approaches to ZTA implementation (identity-centric, microsegmentation-based, software-defined perimeter, and cloud-native) and evaluated selected Zero Trust mechanisms in a virtualized laboratory. A baseline Flask application without access control was compared with a protected prototype that used HTTP Basic Authentication, static role-based access control, Nginx with HTTPS, and threshold-based log monitoring. The prototype rejected both predefined unauthorized access scenarios, whereas the baseline rejected neither. In a 100-request test, mean latency was 1.759 ms for the baseline and 1.863 ms for the protected application; mean resident memory was 37.265 MB and 37.809 MB, respectively. The monitoring rule generated alerts in all three threshold-triggering scenarios but could not distinguish incorrect passwords from potentially suspicious activity. The results support the feasibility of evaluating selected Zero Trust mechanisms on a resource-constrained virtual machine. They do not establish the effectiveness of a complete enterprise ZTA.

Index Terms— Zero Trust Architecture; authentication; RBAC; virtualization; security monitoring; performance evaluation.

I. INTRODUCTION

The adoption of cloud computing, distributed applications, and remote access has increased the complexity of protecting enterprise resources. Traditional perimeter-oriented security places substantial emphasis on controlling access to a trusted internal network. However, network location alone cannot reliably establish whether a requesting user, device, or application should be trusted.

Zero Trust Architecture (ZTA) addresses this limitation by requiring resource-specific access decisions rather than implicit trust based on network position. NIST SP 800-207 describes a policy-driven architecture in which authentication, authorization, and contextual information contribute to access decisions [1]. The BeyondCorp initiative likewise illustrated a shift toward access decisions informed by user and device identity rather than physical network location [2].

Zero Trust is not a single configuration. Implementations may combine identity management, least-privilege authorization, microsegmentation, encrypted communication, policy enforcement, and monitoring [1], [3]. As a result, the practical design and operational consequences of ZTA vary by environment.

Buck et al. identified gaps in empirical understanding of Zero Trust benefits and implementation trade-offs [4]. Although architectural guidance is available, the measurable behavior of individual controls depends on workload and deployment conditions. This study therefore combines a thematic comparison of architectural approaches with a controlled component-level experiment.

The study addresses three research questions:

1. RQ1: What are the major approaches to implementing ZTA, and how do their security mechanisms and deployment requirements differ?

2. RQ2: How does implementing selected Zero Trust mechanisms affect access-control enforcement, response latency, and resource consumption in a virtualized environment?

3. RQ3: How effectively does threshold-based monitoring identify repeated authentication failures, and what limitations emerge from its detection logic?

The contribution is a comparative synthesis of architectural approaches and a reproducible evaluation of selected access-control, transport-security, and monitoring mechanisms. The prototype is not presented as a complete enterprise Zero Trust deployment. The following sections review the literature, describe the laboratory methodology, report measured results, and discuss their interpretation and limitations.

II. LITERATURE REVIEW

2.1. Fundamental Principles of Zero Trust Architecture

NIST SP 800-207 defines Zero Trust as a resource-oriented security approach that does not grant implicit trust based on network location [1]. Access to resources should be evaluated according to policy and relevant information about the requesting subject, device, and environment.

The NIST logical architecture distinguishes a Policy Engine (PE), which determines access decisions; a Policy Administrator (PA), which coordinates authorized communication; and a Policy Enforcement Point (PEP), which enables or blocks communication [1]. These are conceptual functions and need not correspond to three distinct software products.

Core principles include least-privilege access, per-session decisions, protected communications, and the collection of security-relevant information. Syed et al. survey related mechanisms, including authentication, authorization, microsegmentation, and security automation [3]. Deploying one of these controls alone does not constitute a complete ZTA.

2.2. Architectural Approaches to ZTA Implementation

2.2.1. Identity-Centric Approach

Identity-centric approaches emphasize verification of users, devices, and other entities, followed by authorization based on policy. RBAC assigns permissions to roles rather than to

individual users [9], [10], whereas ABAC evaluates attributes of the subject, resource, and environment [11]; both can enforce differentiated privileges. BeyondCorp provides a practical example of identity- and device-informed enterprise access [2]. NIST describes enhanced identity governance as an important deployment approach [1].

2.2.2. Microsegmentation-Based Approach

Microsegmentation divides infrastructure into smaller security boundaries and restricts unnecessary communication between workloads. NIST identifies microsegmentation as one approach to Zero Trust deployment [1]. Its operational challenges include maintaining granular policies and understanding legitimate traffic flows [3]. Binding a service to the loopback interface restricts exposure but is not equivalent to multi-host microsegmentation. CISA's 2025 guidance likewise recommends a phased approach to planning and deploying microsegmentation [7].

2.2.3. Software-Defined Perimeter

Software-Defined Perimeter (SDP) approaches use policy-controlled connectivity to restrict which subjects can establish communication with protected resources. The CSA SDP specification describes a control plane, in which a controller authorizes connections, and a data plane, in which connectivity between initiating hosts and accepting hosts is enforced [12]. Gateways, agents, and controllers may participate in enforcement, and the specification presents SDP as an architecture that supports Zero Trust principles [12]. NIST guidance also discusses controlled access models and deployment examples [1], [5]. A conventional reverse proxy alone should not be described as a complete SDP; the Nginx reverse proxy in this prototype did not include a controller or client agents.

2.2.4. Cloud-Native and Service-Oriented Approach

Cloud-native deployments must account for application and workload identities across distributed environments. NIST SP 800-207A discusses application-centric access control in cloud-native and multi-cloud systems, including policy enforcement for service interactions [6]. Mechanisms such as workload identity and mutual TLS are relevant to these environments but were not implemented in the present single-VM prototype.

2.3. Existing Research and Gap

Syed et al. provide a broad survey of Zero Trust technologies and challenges [3]. Poirrier et al. offer a more recent taxonomy of zero trust models and architectures and analyze the benefits and drawbacks of migrating toward zero trust [8]. Buck et al. examine academic and practitioner perspectives and highlight gaps in empirical evidence [4]. NIST SP 800-207A extends architectural guidance to cloud-native applications [6]. Together, these works provide a conceptual basis for implementation but do not determine the performance of this study's specific Flask and Nginx configuration.

The present study addresses a deliberately narrow empirical question: whether selected controls enforce predefined policies and what resource and latency differences are observed in a reproducible virtualized laboratory. Its results should not be generalized to enterprise-scale ZTA or treated as a security benchmark across alternative secure architectures.

III. METHODOLOGY

3.1. Research Design

The study combines a thematic comparative literature analysis (RQ1) with a quantitative, controlled laboratory experiment (RQ2 and RQ3). The experiment uses two application configurations: a baseline without authentication or role-specific authorization and a protected prototype implementing those controls. The analysis is descriptive; no statistical significance or population-level security effectiveness is claimed.

3.2. Experimental Environment

The laboratory ran Ubuntu Server ARM64 in a UTM virtual machine on an Apple Silicon MacBook. Python Flask implemented the applications. Nginx provided a network-facing reverse proxy and HTTPS using a laboratory self-signed certificate. Client measurements used curl, while resource measurements used Linux process statistics. The applications used Flask's development server rather than a production application server.

Table 1. Experimental environment and tools.

Component	Implementation	Purpose
Virtual machine	UTM / Ubuntu Server ARM64	Controlled laboratory
Application	Python Flask	Baseline and protected endpoints
Reverse proxy	Nginx	Routing and TLS termination
Authentication	HTTP Basic Authentication	Credential verification
Authorization	Static viewer/admin roles	Least-privilege test cases
Monitoring	Python + Nginx access log	Threshold-based alerts
Measurement	curl and /proc / ps	Latency and resource metrics

3.3. Baseline and Protected Prototype

Both applications exposed /public, /reports, and /admin endpoints. The baseline did not authenticate or authorize requests. The protected application required credentials and used viewer and admin roles. Both roles could access /public and /reports after authentication, while only admin could access /admin. Requests without credentials returned HTTP 401; authenticated viewers requesting /admin received HTTP 403; authorized administrators received HTTP 200. Flask applications were bound to 127.0.0.1, and the protected application was exposed through Nginx. The prototype did not implement MFA, continuous authentication, dynamic trust scoring, multi-host segmentation, or service mesh.

3.4. Experimental Procedure and Metrics

• Access control: three /admin scenarios were evaluated for each configuration: no credentials, viewer credentials, and admin credentials. Outcomes were recorded as HTTP status codes.

• Local latency: 30 sequential curl requests per configuration were measured in a first experiment. A separate resource test executed 100 requests per configuration and recorded request

latency, HTTP status, resident memory, and process CPU percentage.

• CPU time: accumulated process user and system CPU time was compared over 300 successful requests per configuration using Linux process statistics.

• Transport performance: an initial experiment measured 30 HTTP requests followed by 30 HTTPS requests through Nginx. A second experiment alternated protocol order over 30 rounds (30 requests per protocol). Each curl invocation included connection establishment; measurements therefore do not isolate TLS encryption cost.

• Monitoring: a Python script inspected Nginx access logs and generated an alert after five HTTP 401/403 responses from one IP address within 60 seconds. Tests included two unauthorized bursts, one incorrect-password burst, a successful-request series, and a below-threshold series.

3.5. Analysis

The analysis uses counts, arithmetic means, medians, and sample standard deviations where available. Percentage change is calculated as ((ZTA − Baseline) / Baseline) × 100. The datasets are not pooled across distinct experiments. Monitoring delays are approximate because source log timestamps have second-level resolution. The resource-test configurations were measured sequentially, creating possible ordering and background-load confounds.

IV. RESULTS

IV. 1. ARCHITECTURAL COMPARISON

The literature analysis identified identity-centric, microsegmentation-based, SDP, and cloud-native/service-oriented approaches as useful comparison categories (Table 2). Their emphasis differs across identity, connectivity, workload isolation, and distributed service policy. The approaches also overlap in practice, since a real deployment typically combines several of them [1], [8]. The experimental prototype implemented mainly identity-centric controls, alongside TLS and basic log monitoring.

Table 2. Comparison of ZTA implementation approaches.

Approach	Primary focus	Typical mechanisms	Implementation challenge
Identity-centric	Who may access a resource	IAM, MFA, RBAC/ABAC	Identity and policy lifecycle
Microsegmentation	Which workloads may communicate	Network policies, segmentation	Maintaining granular rules
SDP	When connectivity is established	Agents, gateways, policy controls	Additional coordination
Cloud-native	Service and workload identity	mTLS, workload identity, service	Distributed policy management

The prototype in this study corresponds mainly to the first row, with TLS and basic monitoring added.

4.2. Access-Control Outcomes

Table 3. Access-control results.

Scenario	Baseline status	Protected status	Expected outcome
No credentials → /admin	200	401	Denied
Viewer → /admin	200	403	Denied
Admin → /admin	200	200	Allowed

The protected application denied 2 of 2 predefined unauthorized scenarios; the baseline denied 0 of 2. Both allowed the authorized administrator. These are outcomes for the selected scenarios, not an estimate of overall security effectiveness.

4.3. Local Response Latency

Table 4. Local latency measurements from two distinct experiments.

Metric	Baseline	Protected prototype
Requests (first test)	30	30
Mean latency, ms	0.464	0.398
Median latency, ms	0.428	0.368
Sample SD, ms	0.163	0.087
Requests (resource test)	100	100
Mean latency, ms	1.759	1.863
Median latency, ms	1.700	1.724

The 30-request experiment recorded lower mean latency for the protected prototype. The 100-request resource test recorded an increase of approximately 0.104 ms (5.9%) for the protected prototype. The experiments therefore did not produce a consistent direction of latency difference.

4.4. Resource Consumption and CPU Time

Table 5. Resource and CPU-time measurements.

Metric	Baseline	Protected prototype
Mean resident memory, MB	37.265	37.809
Maximum resident memory, MB	37.312	37.855
Process CPU reported by ps	0.0%	0.0%
Successful CPU-test requests	300/300	300/300
Accumulated CPU time, ms	120	120
CPU time per request, ms	0.400	0.400
Total wall time, ms	831.380	830.028

The mean resident-memory difference was 0.544 MB (approximately 1.46%). CPU-time values were equal at the available measurement resolution. The 0.0% process CPU reports do not demonstrate zero processing cost.

4.5. HTTP and HTTPS

Table 6. Alternating-order HTTP/HTTPS comparison through Nginx.

Metric	HTTP	HTTPS
Requests	30	30
Successful responses	30	30
Mean latency, ms	4.766	11.212
Median latency, ms	3.968	11.108
Sample SD, ms	3.989	2.028

Mean HTTPS response time exceeded HTTP by approximately 6.446 ms (135.3%). These curl measurements included fresh connection establishment and, for HTTPS, TLS handshake processing; they do not measure pure

cryptographic overhead or application-level authentication cost.

4.6. Monitoring Outcomes

Table 7. Threshold-based monitoring results.

Scenario	Requests	Status	Alert	Reported delay
Unauthorized burst 1	5	401	Yes	741 ms
Unauthorized burst 2	5	401	Yes	112 ms
Incorrect password	5	401	Yes	916 ms
Authorized requests	20	200	No	N/A
Below threshold	4	401	No	N/A

Three threshold-triggering scenarios generated alerts, and neither the successful-request series nor the below-threshold series generated an alert. Mean reported delay across the three alerts was approximately 590 ms. The monitoring rule did not distinguish benign repeated password errors from potentially suspicious access attempts. Reported millisecond delays are approximate because log event timestamps had only second-level precision.

V. DISCUSSION & CONCLUSION

5.1. Interpretation of Findings

The clearest functional result concerns access control: adding authentication and static RBAC changed two predefined unauthorized /admin requests from HTTP 200 to HTTP 401/403 while preserving administrator access. This supports the practical value of explicit authorization for these scenarios, consistent with the principles of NIST SP 800-207 [1].

Performance observations were less conclusive. The 100-request test showed modest increases in mean latency and memory, whereas the separate 30-request test showed a lower mean latency for the protected application. Sequential execution, short request times, process scheduling, and Flask development-server behavior may explain part of the variability. These measurements should not be interpreted as precise causal estimates of authentication overhead.

HTTPS exhibited higher total response time than HTTP in the alternating-order test, but the difference includes connection and handshake costs. Monitoring generated alerts for configured failure bursts, but its static threshold did not infer user intent. Consequently, the system demonstrated event detection, not comprehensive threat detection.

5.2. Limitations

First, all application components ran within one virtual machine. Restricting Flask to loopback was not equivalent to multi-host microsegmentation or cloud deployment. Second, the prototype used predefined Basic Authentication credentials rather than MFA, adaptive access policies, or continuous trust evaluation. Third, the baseline was deliberately insecure, so the experiment does not compare ZTA against a mature alternative security architecture.

Fourth, sample sizes were limited, workloads were lightweight, and some tests ran sequentially. CPU percentage and CPU-time measurement resolution limited interpretation. Fifth, monitoring used IP-based HTTP status thresholds, and approximate alert delays were derived from second-resolution access-log timestamps.

5.3. Conclusion

This study compared major architectural approaches to Zero Trust and evaluated selected mechanisms in a resource-constrained virtualized laboratory. The literature synthesis distinguished identity-centric, microsegmentation, SDP, and cloud-native approaches while recognizing their overlap. In the experimental component, authentication and RBAC enforced the tested access policy, memory differences were small, local latency results varied by workload, HTTPS increased observed total request time, and threshold-based monitoring detected repeated failures according to its configured rule. These results establish component-level feasibility and measured trade-offs in this laboratory, not comprehensive ZTA effectiveness. They provide a reproducible basis for extending the experiment toward stronger identities, network segmentation, and distributed policy enforcement.

REFERENCES

[1] S. Rose, O. Borchert, S. Mitchell, and S. Connelly, “Zero Trust Architecture,” NIST SP 800-207, 2020. https://doi.org/10.6028/NIST.SP.800-207 (https://doi.org/10.6028/NIST.SP.800-207)

[2] R. Ward and B. Beyer, “BeyondCorp: A New Approach to Enterprise Security,” ;login:, vol. 39, no. 6, pp. 6–11, 2014. https://research.google/pubs/beyondcorp-a-new-approach-to-enterprise-security/

[3] N. F. Syed et al., “Zero Trust Architecture (ZTA): A Comprehensive Survey,” IEEE Access, vol. 10, pp. 57143–57179, 2022. https://doi.org/10.1109/ACCESS.2022.3174679 (https://doi.org/10.1109/ACCESS.2022.3174679)

[4] 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 & Security, vol. 110, art. 102436, 2021. https://doi.org/10.1016/j.cose.2021.102436 (https://doi.org/10.1016/j.cose.2021.102436)

[5] National Institute of Standards and Technology, “Implementing a Zero Trust Architecture,” NIST SP 1800-35, 2025. https://doi.org/10.6028/NIST.SP.1800-35 (https://doi.org/10.6028/NIST.SP.1800-35)

[6] R. Chandramouli and Z. Butcher, “A Zero Trust Architecture Model for Access Control in Cloud-Native Applications in Multi-Cloud Environments,” NIST SP 800-207A, 2023. https://doi.org/10.6028/NIST.SP.800-207A (https://doi.org/10.6028/NIST.SP.800-207A)

[7] Cybersecurity and Infrastructure Security Agency, “The Journey to Zero Trust: Microsegmentation in Zero Trust, Part One: Introduction and Planning,” Version 1.0, July 29, 2025. https://www.cisa.gov/sites/default/files/2025-07/ZT-Microsegmentation-Guidance-Part-One_508c.pdf (https://www.cisa.gov/sites/default/files/2025-07/ZT-Microsegmentation-Guidance-Part-One_508c.pdf)

[8] A. Poirrier, L. Cailleux, and T. H. Clausen, “Is Trust Misplaced? A Zero-Trust Survey,” Proceedings of the IEEE, vol. 113, no. 1, pp. 5–39, Jan. 2025. 10.1109/JPROC.2025.3555131 (https://doi.org/10.1109/JPROC.2025.3555131)

[9]D. F. Ferraiolo and D. R. Kuhn, “Role-Based Access Controls,” 15th National Computer Security Conference,1992.https://csrc.nist.gov/files/pubs/conference/1992/10/13/rolebased-access-controls/final/docs/ferraiolo-kuhn-92.pdf (https://csrc.nist.gov/files/pubs/conference/1992/10/13/rolebased-access-controls/final/docs/ferraiolo-kuhn-92.pdf)

[10] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, “Role-Based Access Control Models,” IEEE Computer, vol. 29, no. 2, pp. 38–47, 1996.10.1109/2.485845 (https://ieeexplore.ieee.org/document/485845)

[11]V. C. Hu et al., “Guide to Attribute Based Access Control (ABAC) Definition and Considerations,” NIST SP 800-162, 2014. https://doi.org/10.6028/NIST.SP.800-162 (https://doi.org/10.6028/NIST.SP.800-162)

[12] Cloud Security Alliance, SDP and Zero Trust Working Group, “Software-Defined Perimeter (SDP)Specification v2.0,”2022. https://cloudsecurityalliance.org/resources/software-defined-perimeter-sdp-specification-v2-0 (https://cloudsecurityalliance.org/resources/software-defined-perimeter-sdp-specification-v2-0)